**Bsc in Computer Science**

# HashMon - Hashrate Monitor Tool

Monitoring tool for Bitcoin mining farms

**Authors**

Birkir Kárason

Grétar Örn Hjartarson

Helgi Rúnar Jóhannesson

Kristmann Ingi Kristjánsson

**Instructor**

Birgir Kristmannsson

**Examiner**

Björgvin Sigurðsson

DEPARTMENT OF COMPUTER SCIENCE
MAY 2020

## Contents

## Introduction

HashMon is a Bitcoin hashrate monitoring tool created in collaboration with Advania Datacenters. Its purpose is to offer customers and employees of Advania Datacenters detailed information about their whole bitcoin farm, all on one website.

Advania Datacenters focuses on 3 different kinds of solutions. One solution is the Bitcoin farm, where they sell customers cheap power, and a place to host their machines. Another one is High-Performance Computing (HPC), where they sell customers compute power to have the ability to process data and perform complex calculations at high speeds. The last one is a Tier 3 data center solution, which is normal data storage with redundant and dual-powered servers.

What is Bitcoin? Bitcoin is a cryptocurrency that is rewarded every 10 minutes or so to workers or miners that work together in a pool on validating transactions that reside within the Bitcoin block. So, every 10 minutes or so we have X many workers(miners) validating all the transactions that occurred while the previous block was being validated.

The problem today is that customers can only see their miners via their pool, and the pool only displays limited data, like the last time the miner was online, and its hashrate. The only way to get detailed information about the miner is to connect to the same network as him and receive the information from him directly by using TCP connections. That’s where we come in. Our solution is to setup a scanner on the networks where the miners reside, and have it scan the network, receive detailed information about all miners on the network and send that data to an API, which will log the data into a Database. We can then use this information to give customers a better overview of what is going on in their farm.

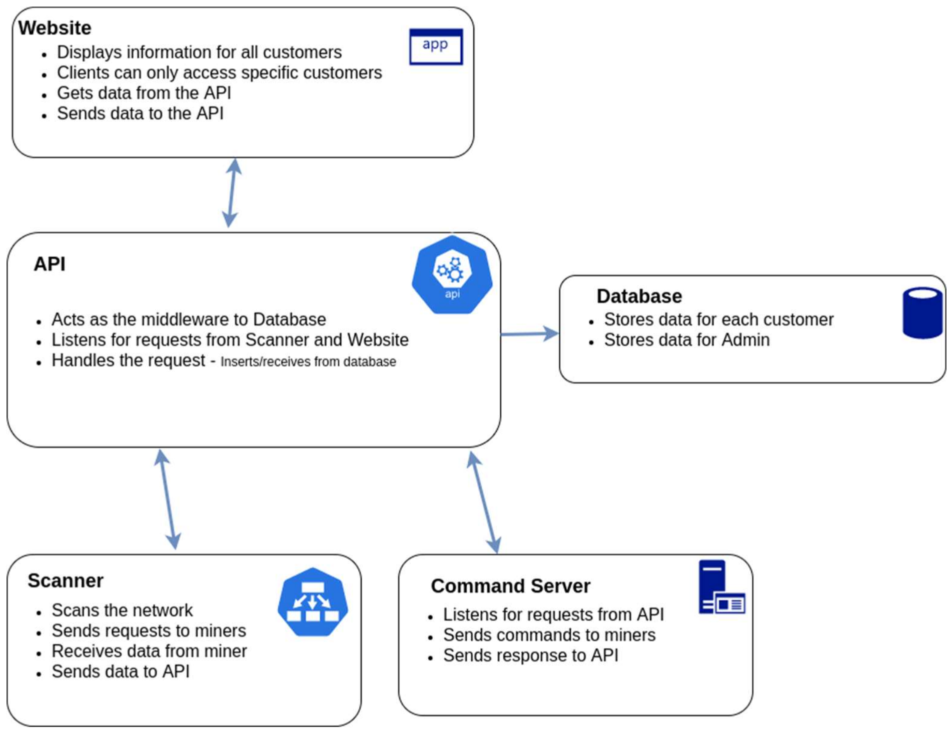
## Project Description

For customers to be able to access detailed information about all their workers we needed to place a device on the networks hosting their miners. This device will pull information from each worker, then we needed to store this information in a database to be able to display it later for the customer. To do this we ended up creating a scanner, API, database and website. As we also need to be able to communicate with each miner to send them commands, we created another application called a command server, which resides on the same device as the scanner.

### System Overview

Below is a diagram explaining the overview of these 5 applications:

1. API which sends and receives information from other applications
2. Scanner that resides within the mining network and receives data from the miners
3. Command server that is hosted on the same machine as the scanner, and listens for requests from the API and sends them to the miners
4. Website that uses the API to get/send information from/to the DB
5. Then we have the Database that stores all our information


Website
Displays information for all customers
Clients can only access specific customers
Gets data from the API
Sends data to the API
app
API
api
Acts as the middleware to Database
Listens for requests from Scanner and Website
Handles the request - Inserts/receives from database
Database
Stores data for each customer
Stores data for Admin
Scanner
Scans the network
Sends requests to miners
Receives data from miner
Sends data to API
Command Server
Listens for requests from API
Sends commands to miners
Sends response to API

## Scanner

The Scanner is an application that resides on a Raspberry Pi within each customer assigned subnet of miners. From within the subnet the scanner locates all the miners on the subnet, and then asks them for information about their current status, such as power consumption, total hashrate and temperature. The scanner then sends the information for each individual miner to the API as well as information for the subnet as a whole (total hash and total power consumption).

## Command server

The command server is an application that resides alongside the scanner on a Raspberry Pi machine. It listens for user commands such as "Restart" and "Configure" for one or more miners and executes them by connecting to each affected miner's internal API and sends the commands as HTTP requests. The "Restart" command quickly reboots the affected miner while the "Configure" command modifies the miner's pool settings.

## Website

The website is designed to display the detailed information that the scanner/API have collected for each customer. Along with displaying information gathered from the miners, the website also allows us to manage their locations, and configuration.  Each customer has his own space within the website where he can manage his farm, he can also view all actions that are performed on the miners in his farm, such as restart miner or configure miner(s).

As we mentioned previously the monitoring tool is not only designed so that each customer can view his farm, but also so that employees of Advania datacenters can view the farms of other customers. To do this we setup the website so that on login you get a list of customers that you have view/write privilege to. Employees could have 1 – n many entries, while customers can only see their own space.

## API

The job of the API is to act as a middleware between applications and the database. Its purpose is to filter out/validate the data of every request before letting it proceed to the database.

## Database

Stores data for all customers, and the admin. It will need to hold data from multiple customers over a long period of time, and the data from multiple customers will be inserted into the database every 5 minutes or so (each scan cycle).

## Project Management

In this project we decided to use the scrum methodology for project management, and at the start of the project we decided to split the team into pairs of two when it came down to design and preparation.

The project was split down into six 2-week sprints, one 4-week sprint, and three 1-week sprints. With one extra 0 sprint in the first week for preparation.

The team was constructed of:

- Product owner: Halldór Þór Helgasson
- Scrum master: Kristmann Ingi Kristjánsson
- Team members: Helgi Rúnar Jóhannesson, Birkir Kárasson and Grétar Örn Hjartarsson

For the first sprint we decided to try out Jira a project tracking software from Atlassian for managing our backlog and excel for our timesheet. But after the sprint we saw that using Jira for the backlog, and excel for timesheet was rather troublesome, since we needed to find a way to export the backlog into excel for our status meeting. So, we ended up using an excel sheet by Birgir Kristmannsson for our project/sprint management, and timesheet.

Since we knew that this project was constructed of 4 core applications, we decided to split the applications between all members so that every application was under development the whole time:

- Kristmann was assigned to the API and the Database
- Birkir and Grétar to the scanner
- Helgi to the website
- Then later Kristmann and Birkir to the command server

Splitting the team like this was a great success, and when the development sprints started things got rolling very fast. Then in the latter part of the project we came together in developing the website.

Kristmann Ingi was the scrum master since this project was his idea after working for Advania Datacenters for about 3 years and seeing a desperate need for these applications in everyday work. His job was mainly to lay out the idea to the whole team and guide them along the way if needed. Also, he made sure that everything that was needed from Advania datacenter was delivered so that we could finish the project on time.

We tried to have weekly status meeting with the product owner, but some weeks that was not possible due to the Covid-19 pandemic.

## Development

In this section we will go over the development phase for each application. This might include design ideas, flow of each request/cycle, frameworks used, or the architecture used for an application.

### Scanner

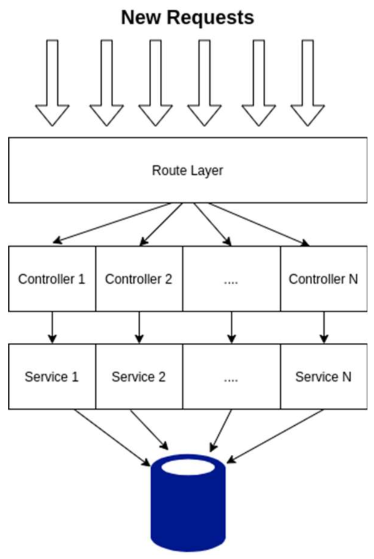
The scanner was developed using Python 3.7. The scanner uses nmap to scan the subnet, and then stores the IP's and MAC's it finds in an array. The scanner then loops through the list of IPs and using a TCP socket, asks for the version of the miner. Depending on the software version of the miner, a sequence of commands is sent to the miner, requesting information. When all commands have been sent, the relevant data is extracted from the responses. After data has been gathered for each miner in the subnet, the data for each miner and the subnet as a whole is sent through HTTP requests to the API. The miner then sleeps for 5 minutes and repeats the process.

### API

The API was developed using Nodejs version 13.14.0, we used the module express to setup middleware to respond to the HTTP requests.

### Architecture

We decided to structure the API as a 3 layered architecture where the incoming requests will be routed to its respective controller, that will validate the data, and send the request forward to the service, then the service will handle the connection to the database. Below is a diagram of the architecture



## Flow of the API

Upon receiving a request, the API will go through this sequence of actions/checks:

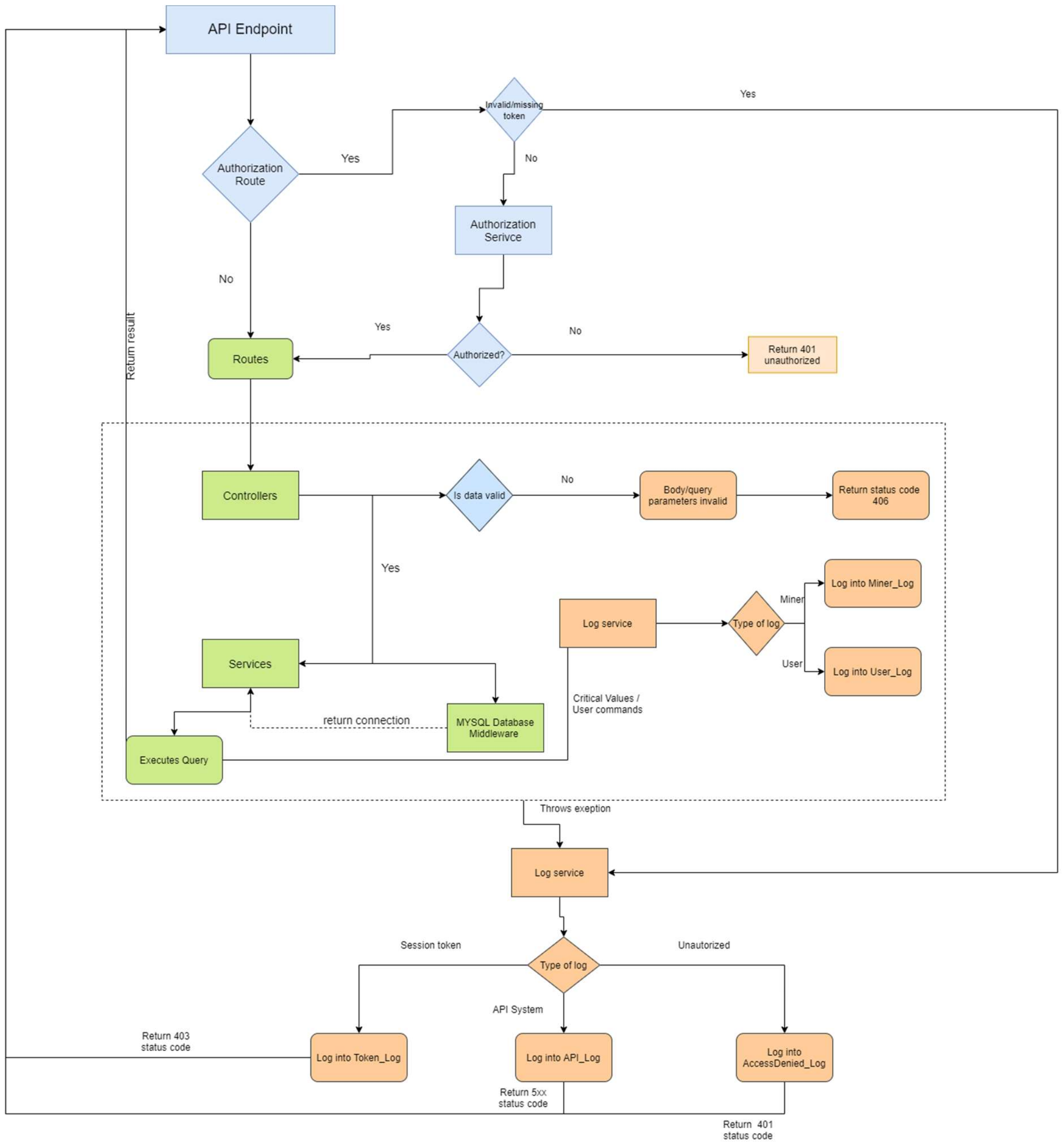
1. An authorization check is performed
    a. If the route requested is an authorization route the session token is validated. Else the route can skip b-e and go straight to (2).
    b. If the token is missing or invalid the request will be logged to the admin database and canceled.
    c. If the token passes the validation it will go through the authorization service where the user is authenticated.
    d. If he doesn't pass the authentication, we send back a 401 (unauthorized) response
    e. If he passes the authentication, the request will proceed to the route requested.

2. The route will then communicate to its respective controller (customer, miner, user, ….)
    a. From the controller the data is validated, for example in a post request expecting a body with property {MAC_Address:"", IP_Address:""} the body will be checked for these 2 properties.
    b. If the body is of the correct format, the controller will call its respective service, if not we will send back a 406 (not acceptable) error to the requestor.

3. In the service each function will expect a user and a customer
    a. The functions will then create a MySQL connection with the given user and customer
        i. User is the requestor and the customer is the database that the user is trying to connect to.
        ii. If the user gets Access Denied to a given database an Access_Denied exception will be thrown.
    b. Then the service will execute the requested query using the connection created in (a) and return the result to the requestor.

4. On exception thrown the API will communicate with the Log_Service which logs the exception into the appropriate Log table.

Below is a flow diagram for the API



## Website

The website was developed using JavaScript and both React (16.12.0) JavaScript library and the Redux JavaScript library. All the code was written by our team and we also used smaller libraries to help with the development which were installed and handled with Node package manager.

We used Camel Casings for all names excluding the names for the components which used Pascal Casing.

The style/theme of the website, which was designed with CSS, was made to look like the main website of Advania data centers, so we picked the same color pattern as the Advania Icon and used that.

We decided that instead of making our own authentication for the login, we would use the MySQL server authentication instead. So, each account information (Name and password) are equal to their login to the MySQL server. We also added an option to login with your Microsoft account, for that we have a database which holds on-to all emails of our users and their login information. To allow users to log in using their Microsoft accounts we added a button which when pressed, opens a pop-up window with a secure Microsoft Login authenticator. We then receive an authenticated object with account information from Microsoft and check it with our database and if the user has an account with us, we log him in.

Icons used on the website are from Font Awesome, the Pie charts are from the react-chartkick library, the line-chart was created using the D3 library.

## Command Server

Like the scanner, the command server was developed using Python 3.7. For incoming connections, we chose Websockets but two methods were used for outgoing connections to miners: a TCP socket and HTTP requests. The command server had to support sending commands to miners manufactured by three different companies: AntMiner, Innosilicon and Whatsminer. When a user command is received the server first has to determine each miner’s manufacturer, which is done by sending a TCP packet with the message “Version” to the miner’s IP address. The miner answers with an object containing its API version, which determines the manufacturer since they all use different API versions.

If the miner is manufactured by Innosilicon then a simple HTTP request is constructed using Basic HTTP authentication and sent to the miners' API. For AntMiner and Whatsminer things got a bit tricky. AntMiner is more secure than the other manufacturers so they use HTTP Digest Authentication for incoming API requests. To be able to construct a Digest authentication header, first we must send a GET request to a specific endpoint and the API responds by sending back a GET request with a Digest header containing the fields Realm (usually host or URI) , Nonce (uniquely generated string) and QOP (quality of protection; authentication mode). These fields along with other values generated by the command server are used to generate a Response field through a series of MD5 hashing that hides the password sent over for authentication. These fields are used to construct a Digest authentication header and a GET request including the header is then sent, which executes the command.

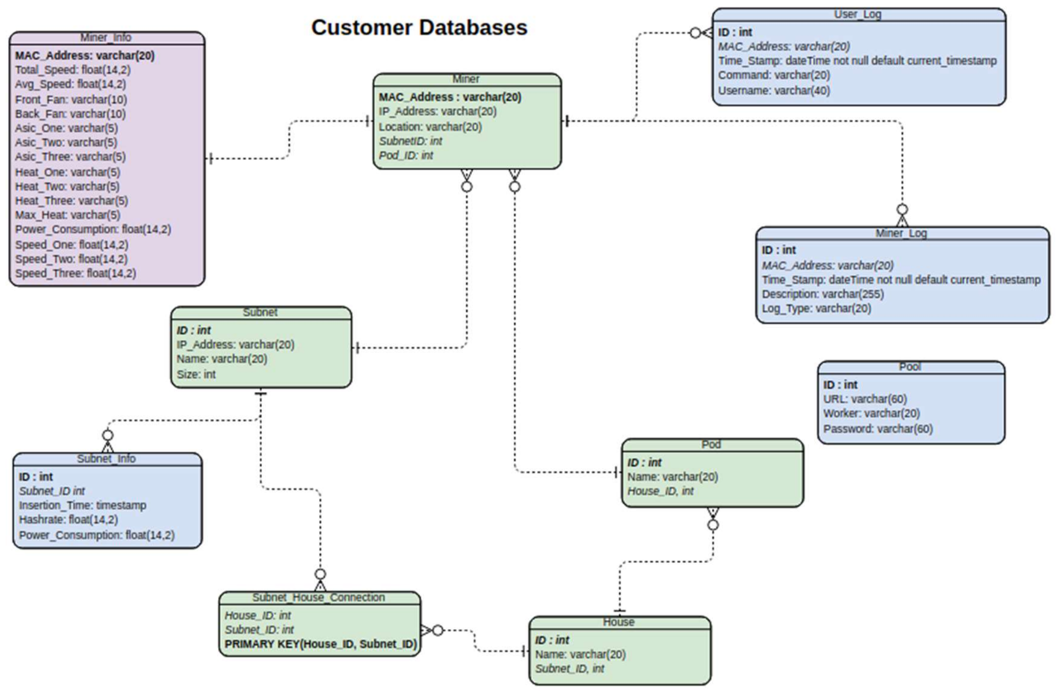
If the miner is manufactured by Whatsminer, we had to keep a HTTP Requests session alive throughout the transaction to store authentication cookies received by logging into the API using a HTTP POST request sent with a body containing the username and password for the miner. Restarting is no problem at this point using a simple GET request since no body is required, but configuring the miner required a lot more thought. To configure a Whatsminer machine we used a POST request containing the configuration for the miner and then we had to search through website metadata, returned by the API after logging in for a security token uniquely generated for each connection and then send another POST request to restart the pool module for the miner so that the changes went through.

## Database

The database was written using MySQL MariaDB version 10.4.12, which Is a community-developed MySQL client. This was decided by the developers at Advania Datacenters.

### Customers

When it came down to how we wanted to manage all of the data that would be stored in our system, we decided that each customer should have his own database instance. This instance was broken down into 10 tables. Miner, Miner_Info, Subnet, Subnet_info, Pool, User_Log, Miner_Log, House, Pod and Subet_House_Connection:



Our opinion was that storing the data like this was a much better solution then just using one big database instance for all clients and having a foreign key reference to which client each data entry belongs to.

### Clients / Customer / Authentication

When it came down to how we wanted to manage the users in our system we decided that all our users will be MySQL users.

- Clients (technicians, admin users) will be users that have read/write privileges on X number of customers.
- Customers (Machine Owners) are basically clients that are the owners of a database and will have full privilege only on their own database. They do not have any privileges on other databases.
- When it comes to authentication we will try to establish a connection to the mysql client and if it’s a success we will send back a JSON session token that we will decode/encode each time the client sends a request for a resource. This way the clients can only access resources according to their privileges, rather than having a user/privileges table in an admin database.

## Testing

### Scanner

We unit-tested the scanner with the Unittest Python library with mocked functions and test data, but the main scanner tests were done by setting up the scanner application on a Raspberry Pi machine and connecting it to a subnet of over 700 miners at Mjölnir in Reykjanesbær where the scanner succeeded effortlessly and sent all the data to our server hosting the API.

### API

To test our API, we decided to use Postman. First, we created a collection of routes where we test various things.  For each route in the collection we then made a test suite that compares the results to expected values. Then we exported this collection as JSON and used it to test our API. We created an external test application in NodeJS that uses the Newman module to read the JSON file exported from postman and runs the tests. We did this about 1000 times with 5 min delay between each cycle, and for each cycle we exported the result to a CSV file.

We then created an external script to read the CSV flies, written in python 3.7.  The script expected a user input for the date to be tested, then it used a directory walk to walk the folder ‘./newman’, and for each file that matched the date given, the script checks for errors within the file, if errors are found we log that into a file, else we delete the file.

### Website

The plan was to have the employees at the Mjölnir data center try the website and test it for us however due to Covid-19 we were not authorized to interact directly with the employees at the Mjölnir data center and could therefore not user test the UI.

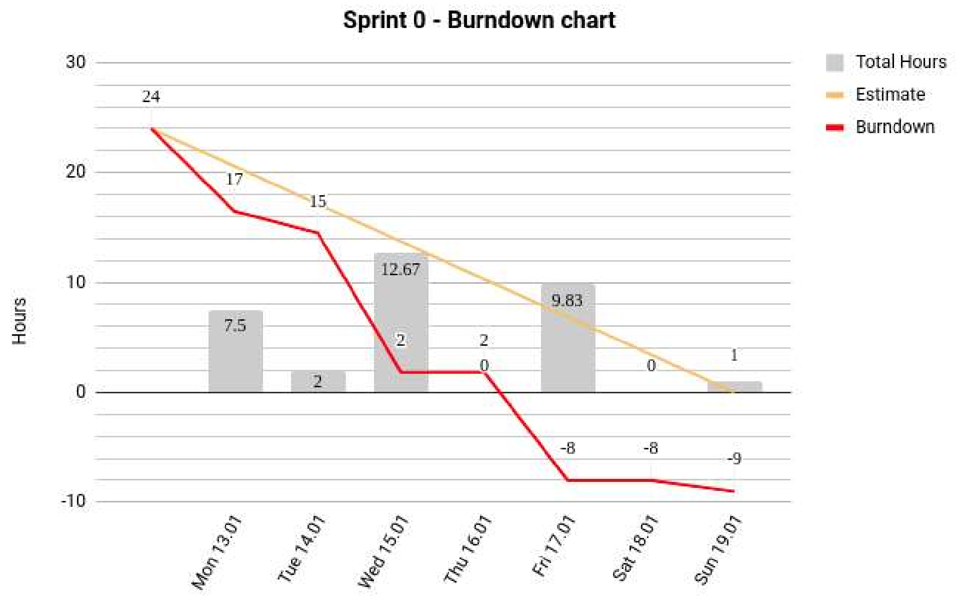
## Progress

### Sprint 0

Date: 9. Jan. - 17. Jan.

Sprint 0 was an extra sprint in the beginning because there were only two of us working on the project (Kristmann and Helgi) and then Birkir and Grétar joined the group on 13. Jan. After forming the full team, Kristmann made a demonstration of the project for the new members.

After the demonstration we made our first draft of the product backlog before consulting with the product owner, and a work agreement (procedural agreement) for the rest of the sprints.



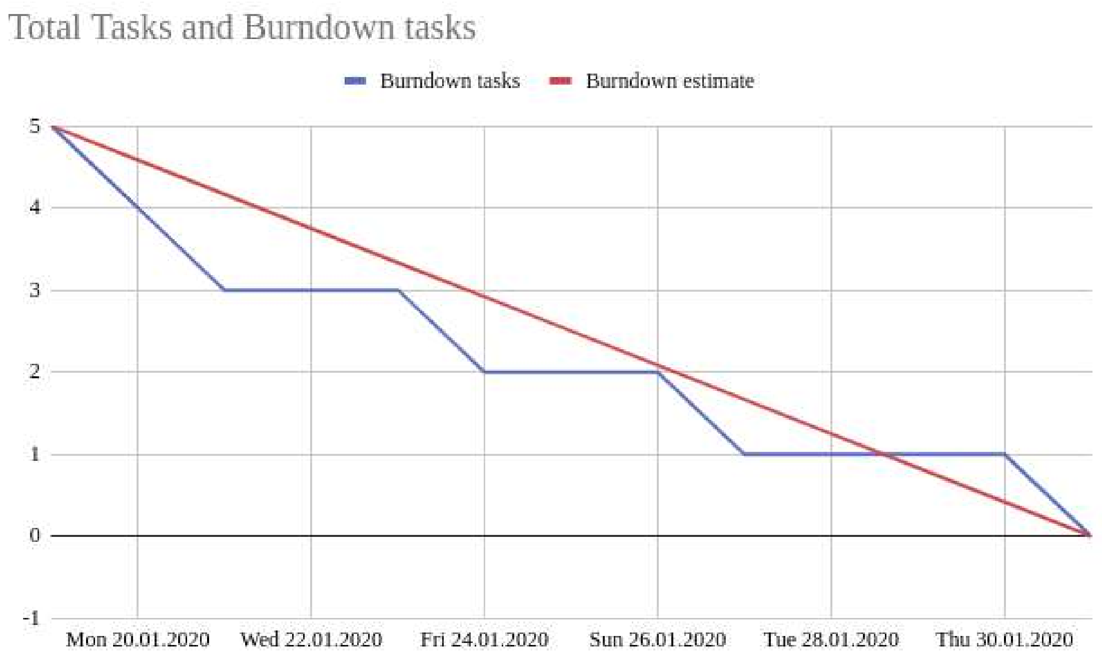
Above is a burndown demonstrating the hours we put into sprint 0. We didn’t include a task burndown since there were only 3 tasks, setting up the backlog, creating a procedural agreement and preparation for the project.

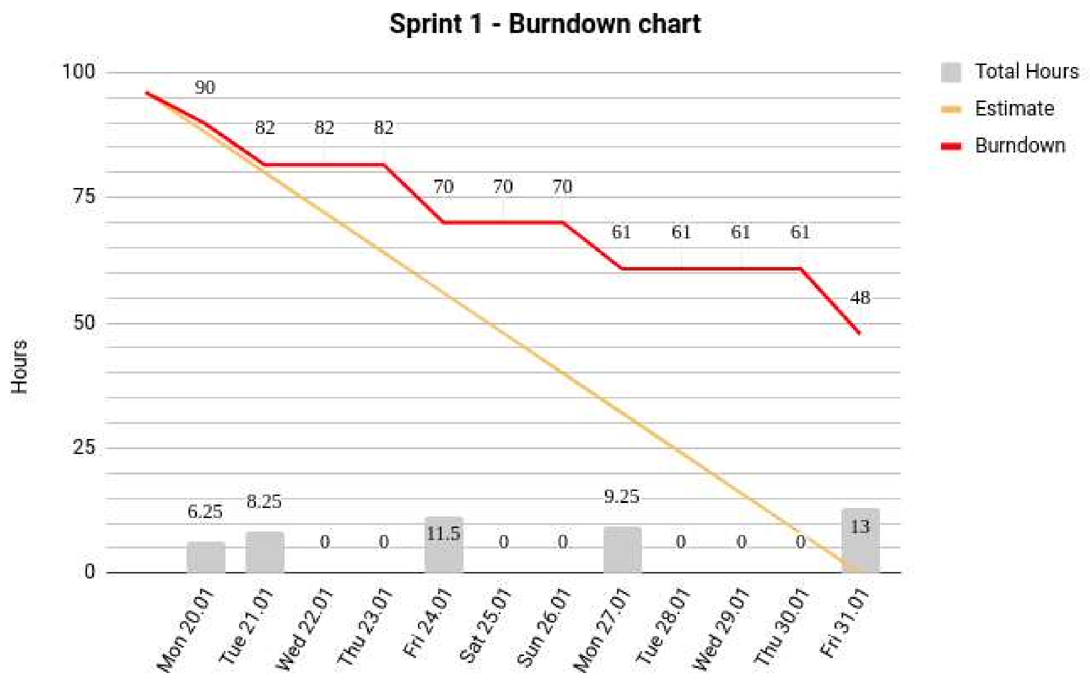
We estimated 24 hours of work for the sprint mainly for preparing the project and getting new team members caught up and we ended up with a total of 33 hours of work in this sprint.

## Sprint 1

Date: 20. Jan. - 31. Jan.

In this sprint our plan was to prioritize our product backlog and make an estimate of the time spent per issue. Then after prioritizing the backlog we picked issues that we found most important to start on, like the design of the whole system.





Above we included two burndown charts, one demonstrating tasks and the other demonstrating time spent during the sprint. In this sprint we fell a bit behind for a couple of reasons:

1. We over-estimated the time some of our issues would take.
2. Birkir was sick most of the sprint.
3. We could not work on an issue for website mock-up designs since the product owner was not available for a meeting to discuss the design so we moved the issue to the next sprint.

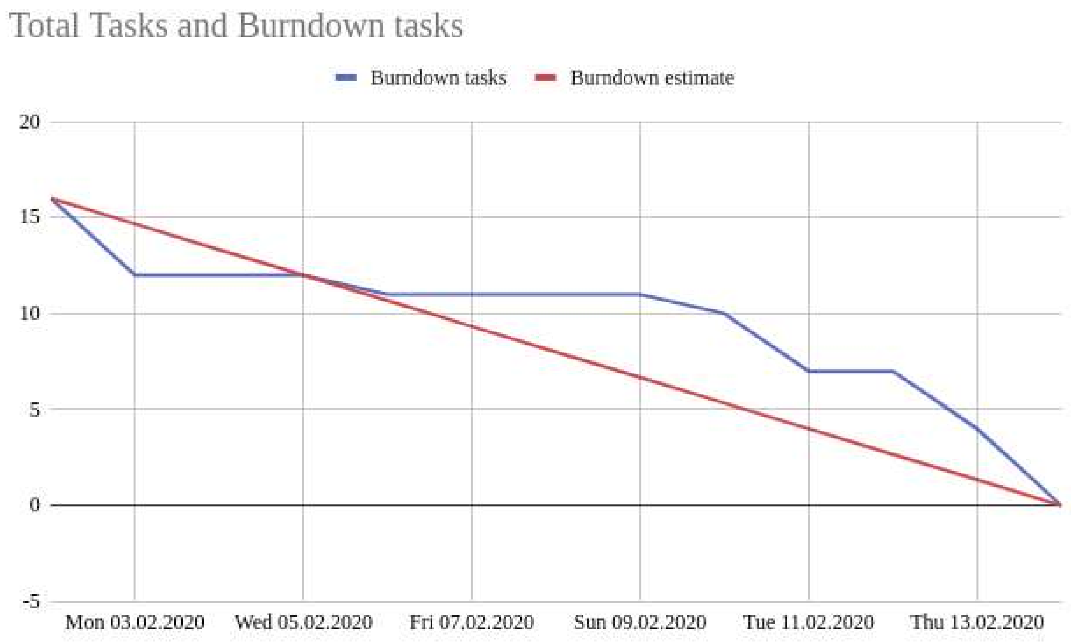
Even though we fell behind we still finished all tasks except for the mock-up task. Out of our estimation of 96 hours we yielded 48 hours of work because of the problems previously stated.

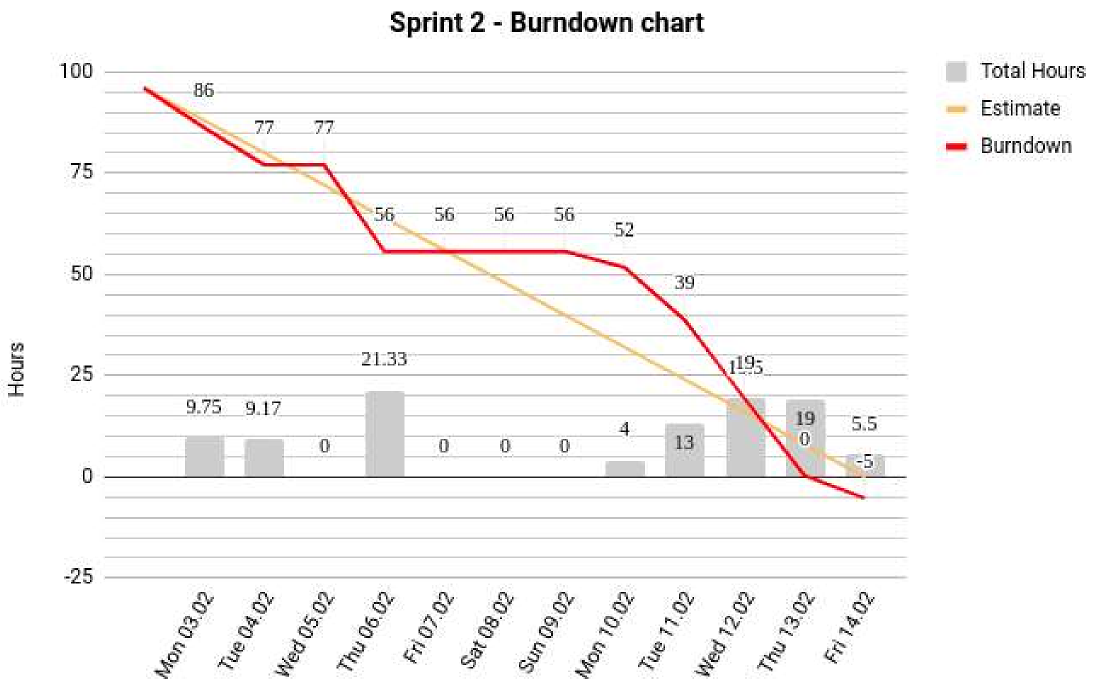
## Sprint 2

Date: 3. Feb. - 14. Feb.

In this sprint our plan was to finish the rest of the design that we could not finish in sprint 1, create the reports for the status meeting on 14. Feb. and prepare for both the status meeting presentation and the peer presentation.

Then in the latter half of the sprint we planned on implementing part of the basic functionality of the scanner, setting up a test API and creating a template for the website.



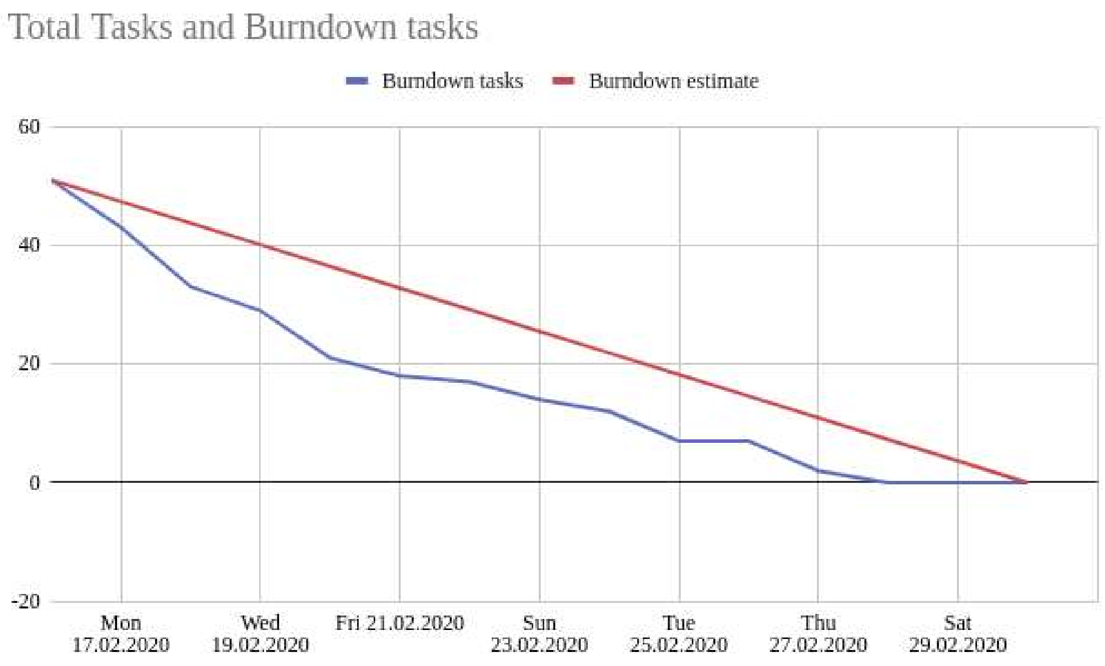


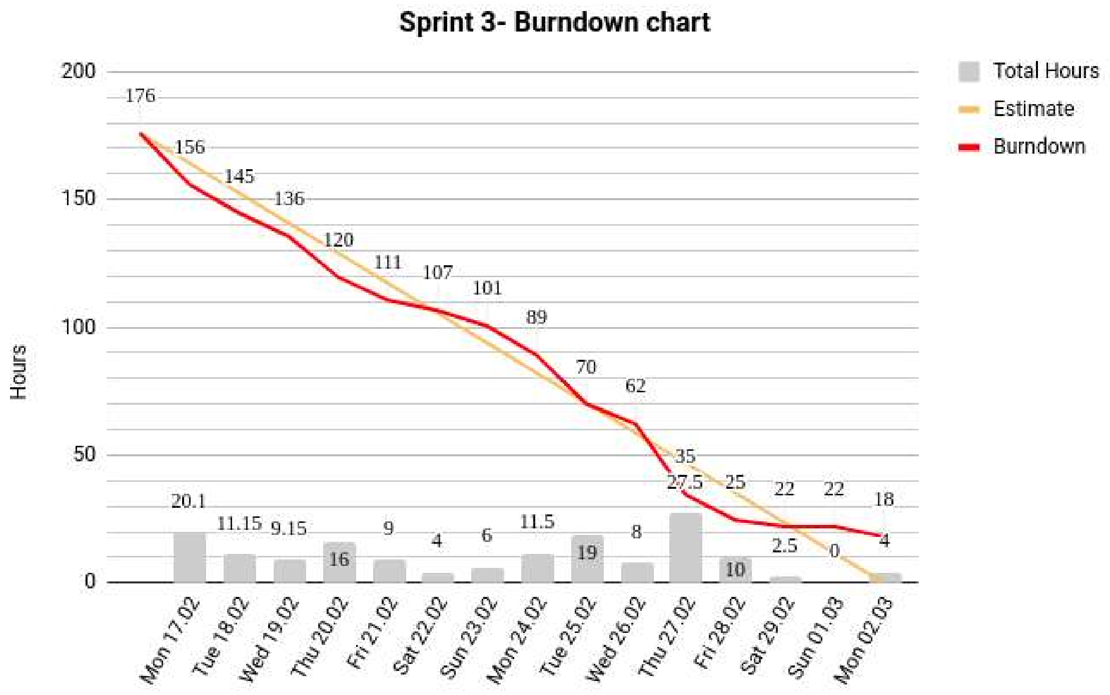
As can be seen in the two burndown charts above, we finished all our tasks for this sprint and ended up spending 5 more hours than we originally estimated.

## Sprint 3

Date: 17. Feb. - 02. March

In this sprint we planned on finishing the higest priority issues for the API, most of the scanner’s base functionality and started doing more regarding the website. We also wanted to finish a sprint and start a new one on Mondays so that we could work on the sprint during the sprints last weekend.



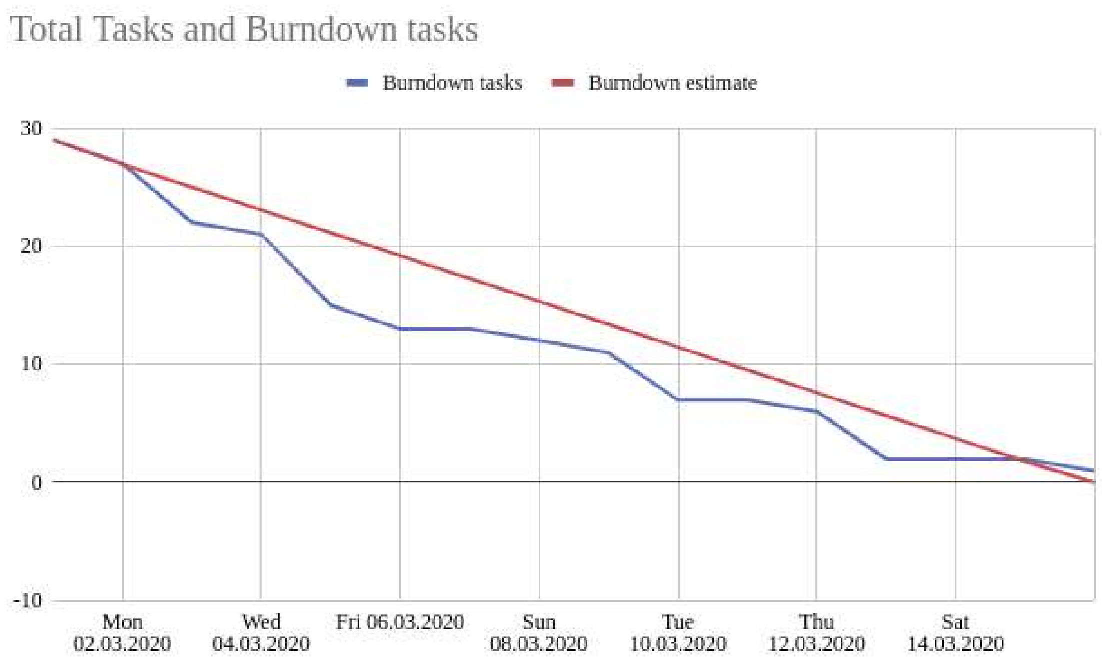


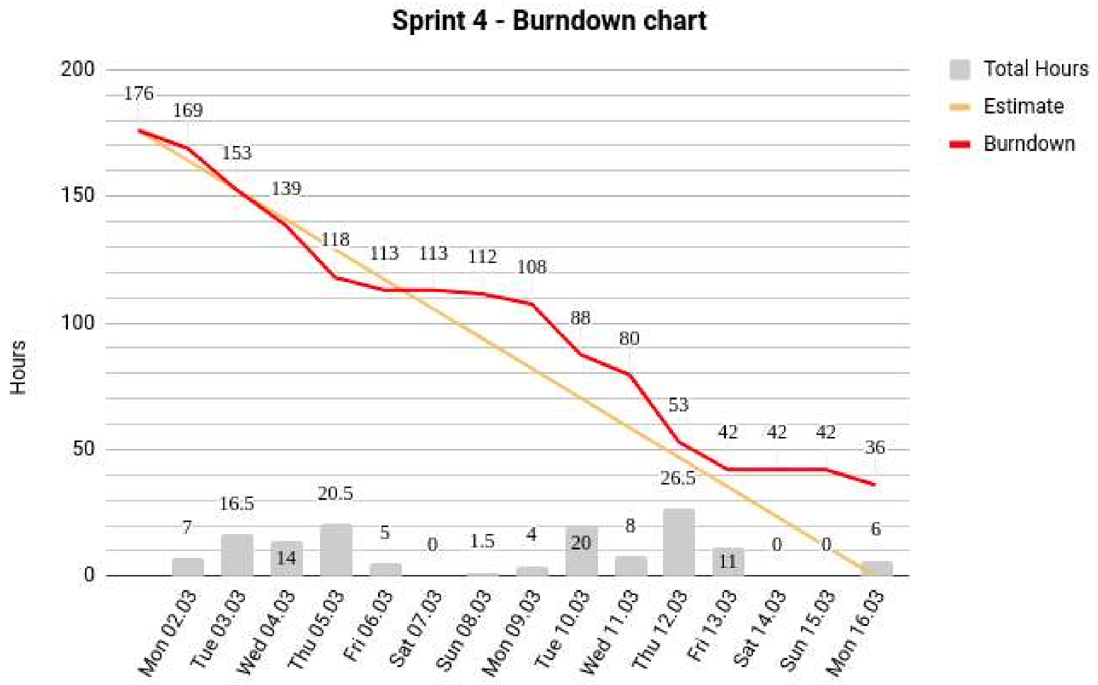
As can be seen in the two burndown charts above, we finished all our tasks for this sprint and ended up spending 18 hours less than we originally estimated.

## Sprint 4

Date: 2. March - 16. March

In this sprint we focused on finishing all the A and B tasks in the scanner, setup the connectivity between the server and our workspace, have a running Raspberry Pi in our workspace that is constantly running the scanner and the functionality on the website. We also started work on the Command server for the scanner (C tasks).



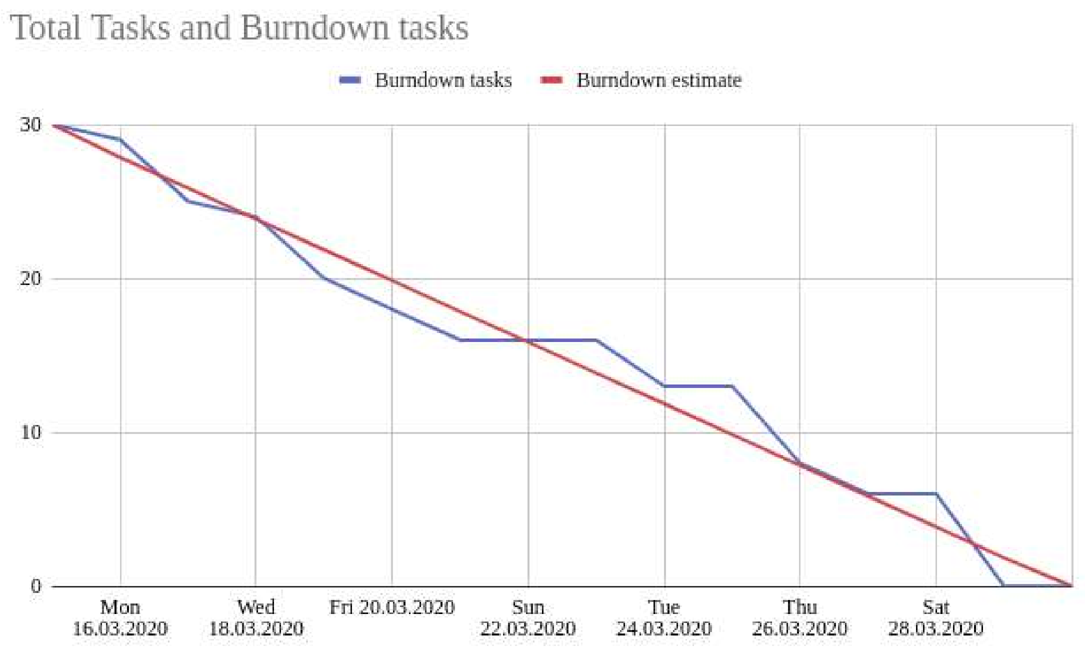


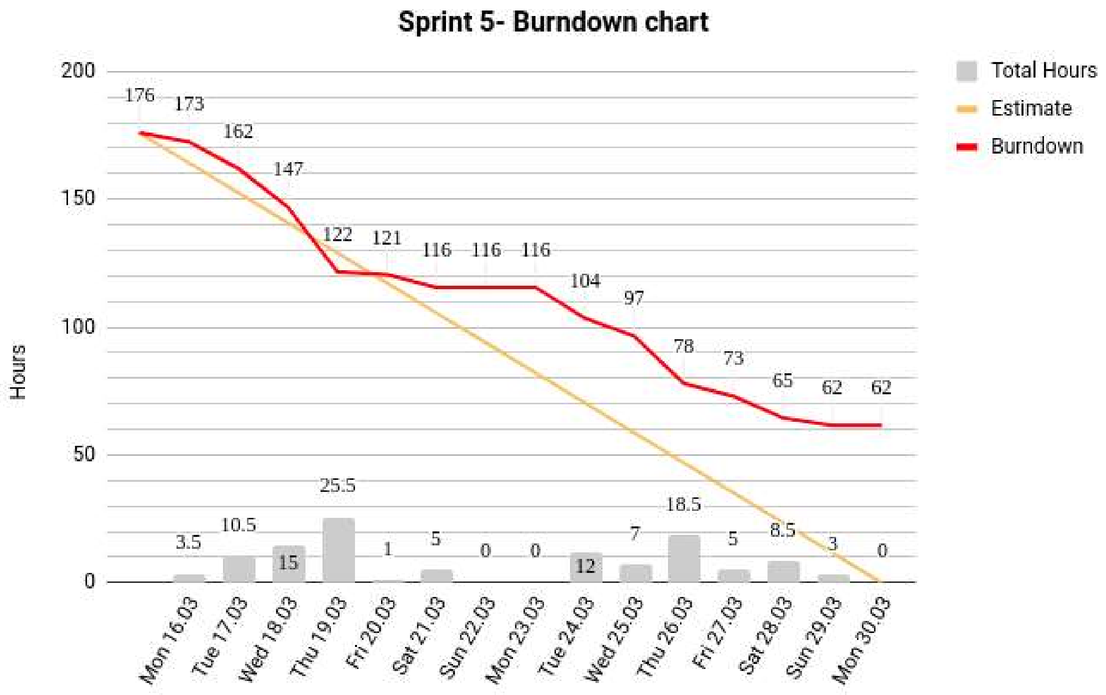
As can be seen in the two burndown charts above, we ended up finishing all our tasks except for one that we couldn't even start which was usability testing on our website. The reason for that was that the workplace of Advania DataCenters is located both in Hafnarfjörður and in Reykjanesbær and because of Covid-19 these 2 workplaces were not allowed to meet each other. The technicians that we planned to interview for the tests are located in Reykjanesbær and our workspace is in Hafnarfjörður.

### Sprint 5

Date: 16. March - 30. March

In this sprint we focused mainly on completing B and C tasks in the API and Website. We also did a little work on the command server and got parts of it working.



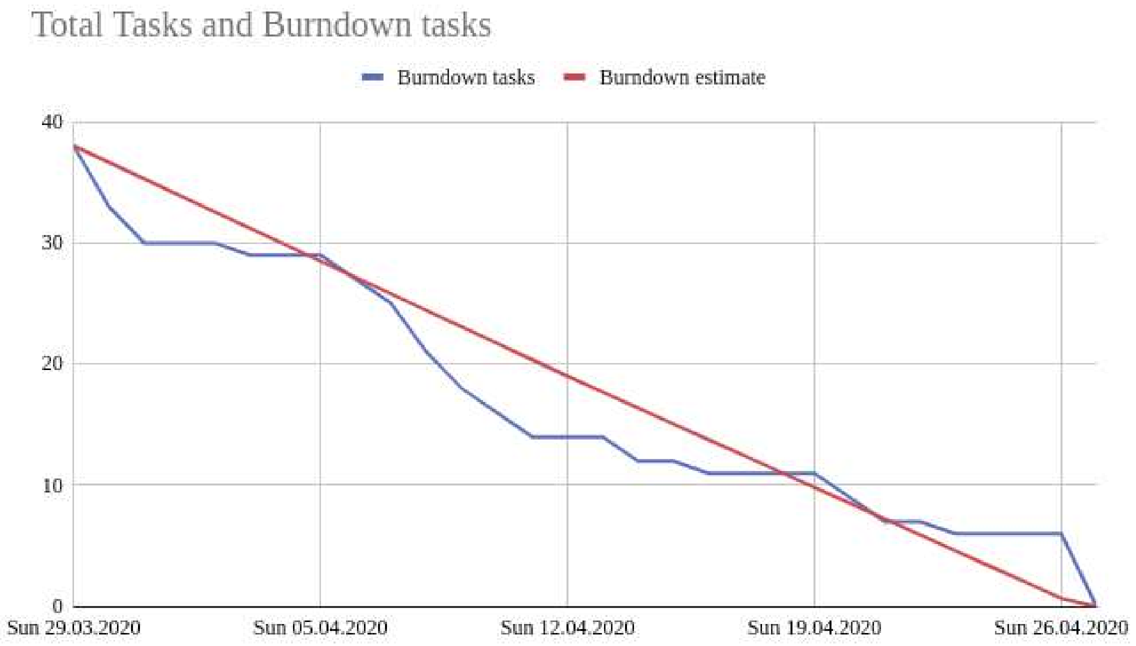


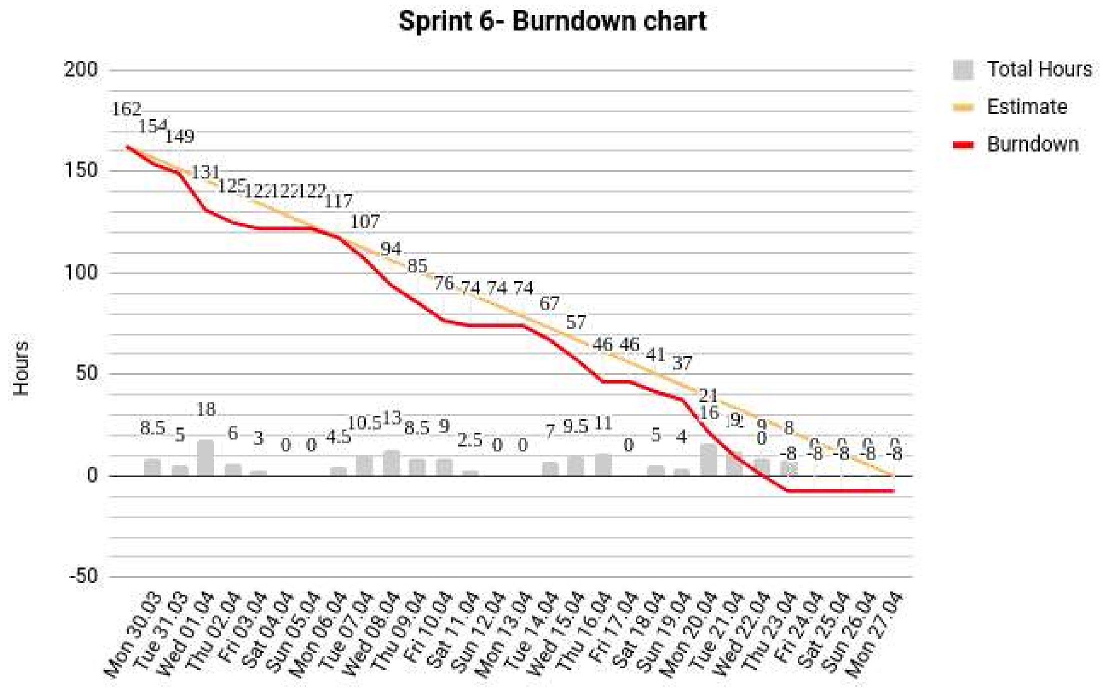
As can be seen in the two burndown charts above, we completed all our tasks, but we overestimated some of them and finished them in shorter timeframe than expected.

## Sprint 6

Date: 30. March -27. April

This sprint was kind of an extra sprint during the exam time. In this sprint all of us were working from home due to covid-19 so the team that was working on the scanner had to join the development of the website since development of the scanner was not possible from home at this point. We mainly focused on styling all the views on the website according to mockup of the site, or just the positioning of the components.



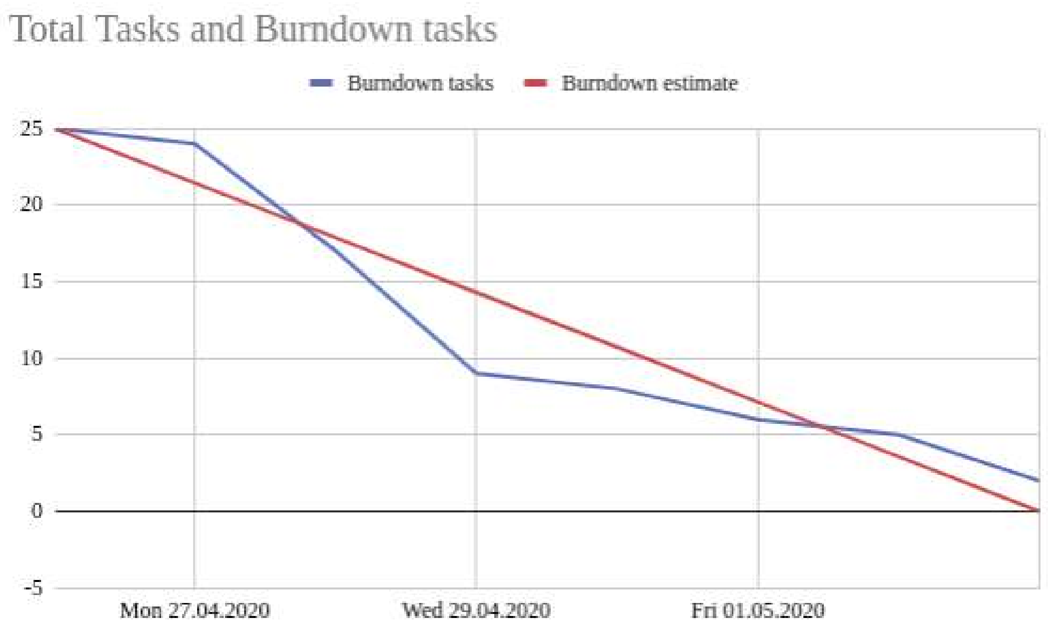


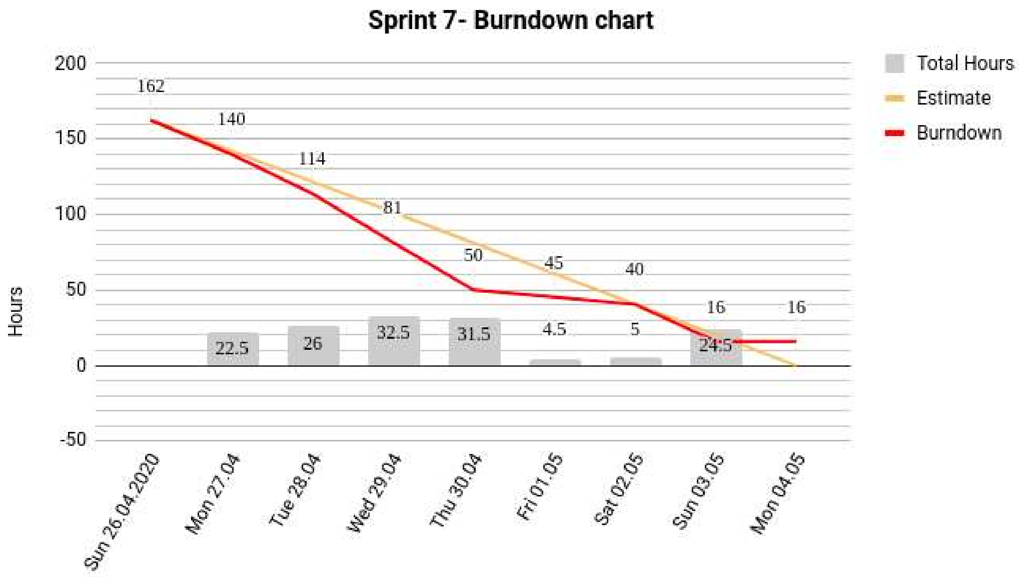
As can be seen in the two burndown charts above, we had an extra number of tasks for this sprint, or 38 and we managed to finish everything in the final day by going over the time estimation by 8 hours.

## Sprint 7

Date: 27. April - 4. May

This sprint was the first sprint of the 3-week period. The main focus in this sprint was to finish the implementation of the functionality. We decided that after this sprint no extra functionality on the website would be added.



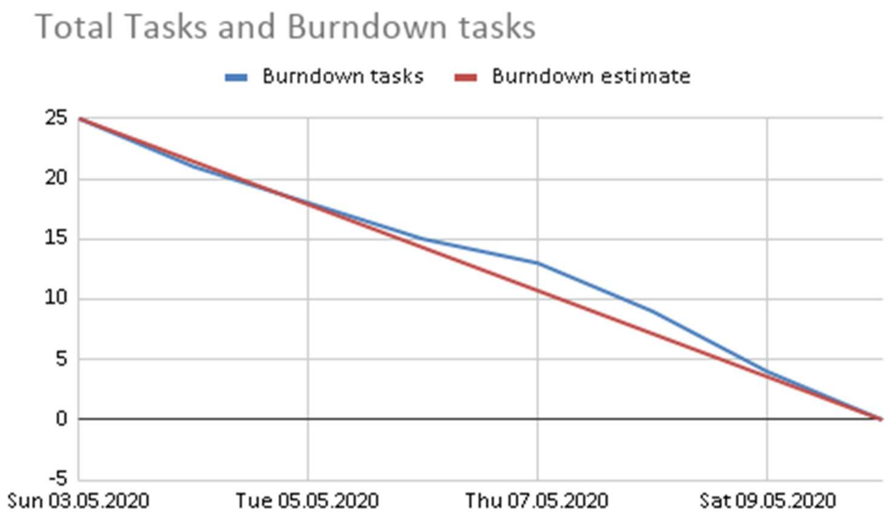


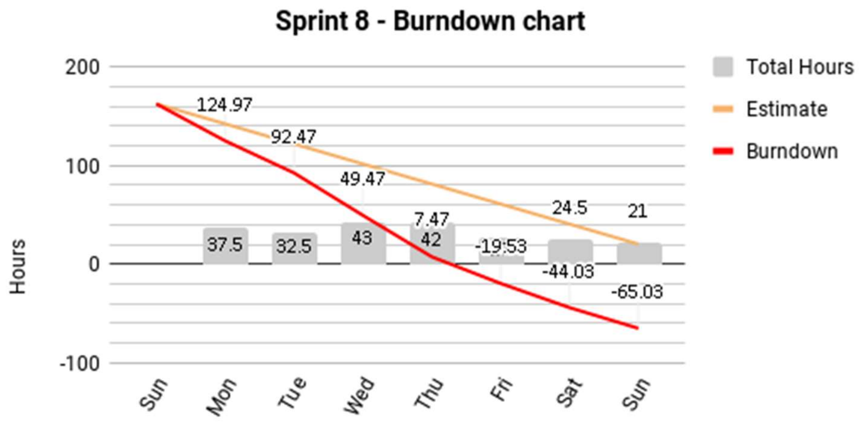
As can be seen in the two burndown charts above, we had set 25 tasks for this sprint, but only managed to finish 23 tasks. The reason for this was that on both Monday and Tuesday one person could not show up with the team, and if we look at the hourly graph, we can see that we were 16 hours behind our estimation.

## Sprint 8

Date: 4. May - 11. May

This sprint was the second sprint of the 3-week period. The main focus in this sprint was to have most of the Final Report finished and try to have no bugs remaining in the website.



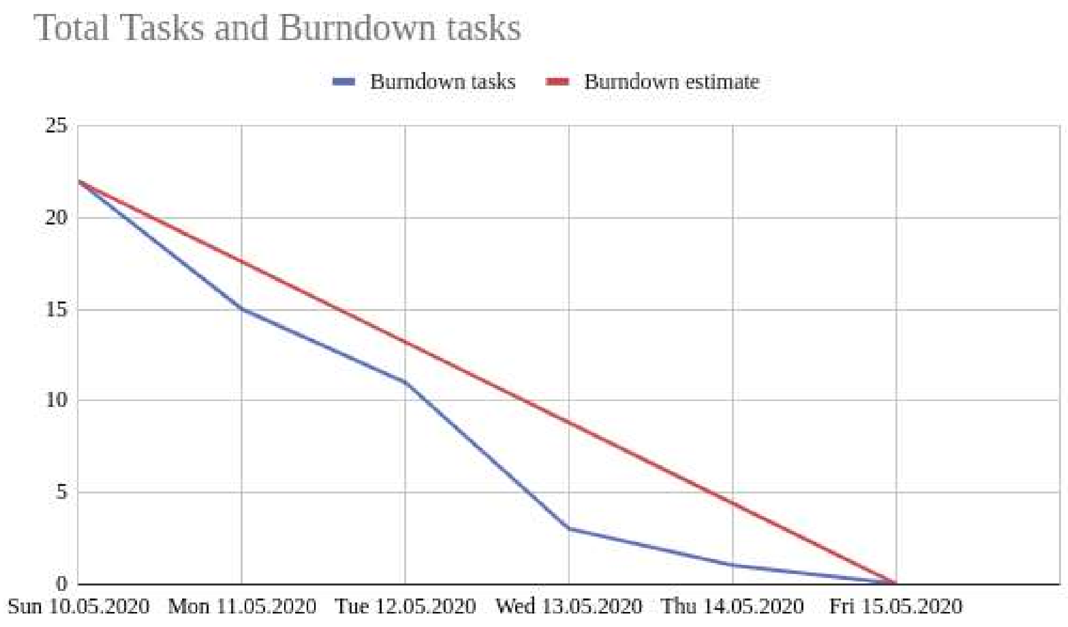


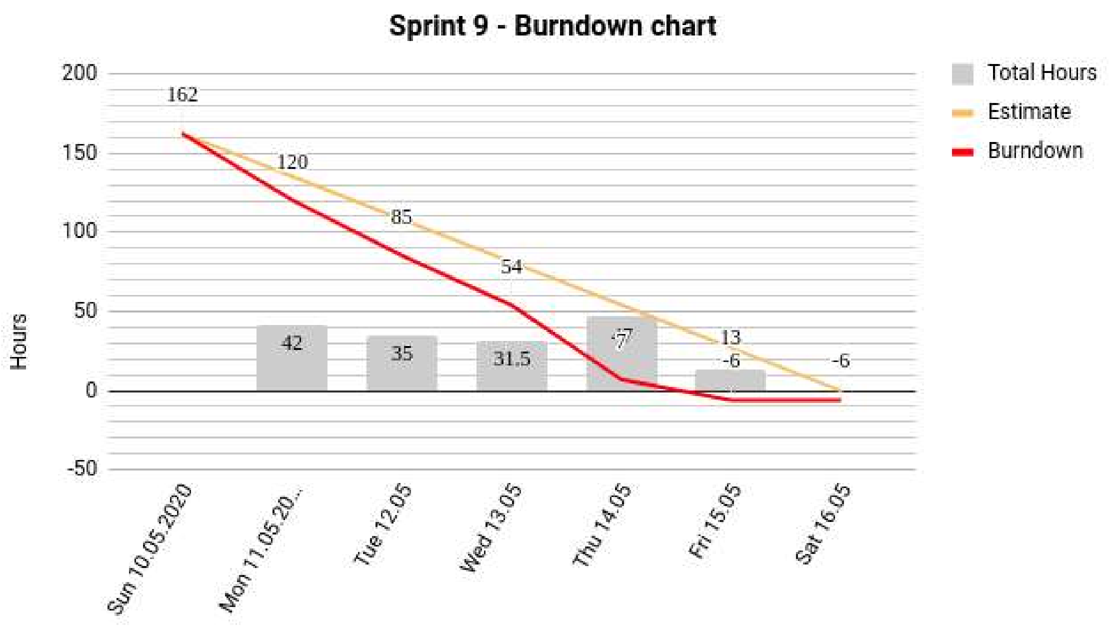
As can be seen in the two burndown charts above, we had set 25 tasks for this sprint and managed to finish 25 tasks with 2 canceled tasks.  The reason we had so many hours assigned to us this week was mostly the minor bugs, we both had to find the bugs and fix them, along with preparing for the third status meeting.

## Sprint 9

Date: 11. May - 15. May

This sprint was the third sprint of the 3-week period. The main focus in this sprint was to have the entire project ready for the final delivery, commenting our code, finishing to style the website and finalizing the Final Report.

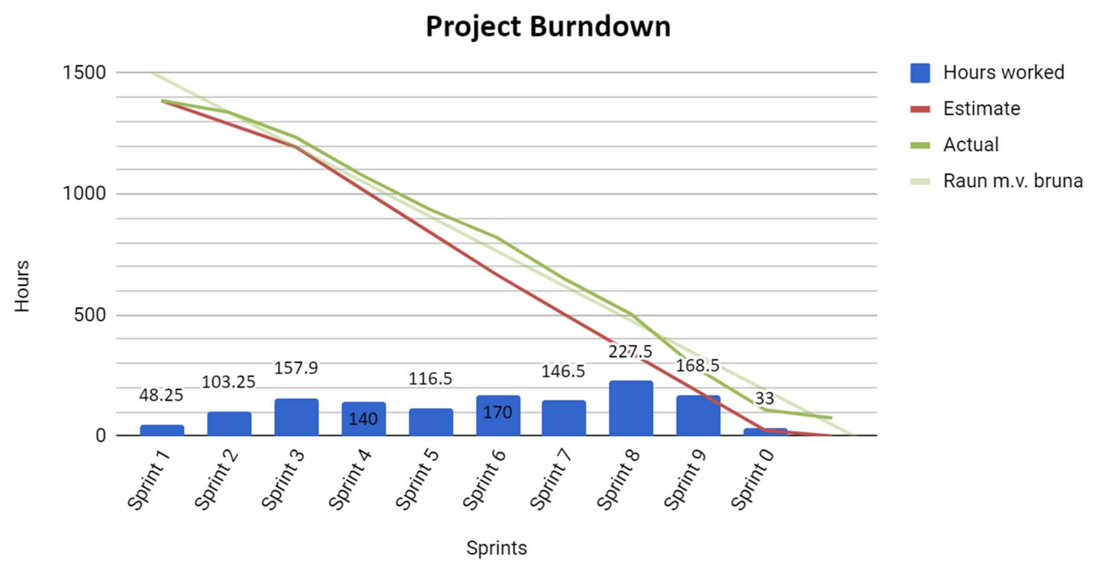




As can be seen in the two burndown charts above, we finished all of our tasks and went over our time estimation by 6 hours.

## Project Burndown

The below graph shows the burndown of the whole project. As can be seen, we did not complete all estimated hours but despite that we finished all high priority issues quickly and started adding on lower priority smaller features.



## Time Breakdown

| | Website | Scanner | API | Database | Reports | Design | Other | Total |
|---|---|---|---|---|---|---|---|---|
| Kristmann | 194 | 34.5 | 119.45 | 9 | 38 | 10 | 86.5 | 491.45 |
| Birkir | 65.5 | 109.45 | 3.5 | 3.5 | 31.33 | 6 | 65.17 | 284.45 |
| Helgi | 170.5 | 0 | 0.5 | 0 | 19.5 | 6 | 58.5 | 255 |
| Grétar | 115 | 40.5 | 0 | 0 | 39.5 | 9 | 72.5 | 276.5 |
| Total | 545 | 123.45 | 151.92 | 12.5 | 184.45 | 31 | 282.67 | 1307.4 |

## Summary

### What went well?

In the beginning of the development we split the group down to cover different parts of the system. Kristmann took on the API and database, Helgi the website and Birkir and Grétar focused on the scanner. This resulted in the development going smoothly and evenly throughout the project and in the end, we joined forces on the website. Overall, communication and teamwork between the team members went very well.

### What could have gone better?

Not a lot of things went wrong with the project itself, but the preparation could have been better. We also could have communicated better during the Covid-19 restrictions especially during the 5th and 6th sprints, where productivity was a bit low. We could have had better attendance during the entire project as well.

### What did we learn?

We learned a lot about the programming languages and frameworks that we used to develop the system, but the biggest lesson was how a system like this is set up, and the connection between its components. It was good to finally use all the knowledge that we have learned during our time at the university to make a whole system that touched on most parts of computer science, and to learn what struggles we might face in the near future.

## What's next?

Even though we managed to get most of the system ready in the 15 weeks we were working on it, it’s safe to say that it won’t be ready for publication.

The first thing on the agenda would be to start the usability testing that we couldn’t perform because of the covid-19 pandemic. After altering the user interface accordingly after inputs from technicians we think that the product should be ready for publication, with few bugs fixed that might come to life.

After getting the product ready for publication our team member Kristmann Ingi will help the deveoplers at Advania Data Centers to setup everything needed to be able to monitor their own machines so they can learn the process of setting up a new customer within the system.

## Appendix

### Backlog

| Priority | Issue | Done | Issue Type |
|---|---|---|---|
| A | As a API i must have the route /api/subnets/:customer/:subnet[POST] that will insert given information into a given subnet | Done | API |
| A | As a API i must have a route /api/subnets/:customer/:subnet[GET] , that returns all entries for a given subnet | Done | API |
| A | As a Test_API i want to have routes so that website can test login locally | Done | API |
| A | Change the test API into a more structured project | Done | API |
| A | Create a MYSQL Database connection middleware | Done | API |
| A | As a API i must have the route /api/miners/:subnet/:customer[GET], that returns all miners within a given subnet | Done | API |
| A | As a Test_API i want to have routes so that scanner can test /Miners/:customer POST route locally | Done | API |
| A | Make a test suite in postman for the route 'api/logs/miner/:customer' | Done | API |
| A | Update the Miner_Info table so it has a timestamp property | Done | API |
| A | Make a test suite in postman for the route '/api/hash/:customer' | Done | API |
| A | Make a test suite in postman for the route '/api/subnets/:customer/:subnet'[get] | Done | API |
| A | Make a test suite in postman for the route '/api/miners/:customer' | Done | API |
| A | Make a test suite in postman for the route '/api/login' | Done | API |
| A | Make a test suite in postman for the route '/api/subnets/:customer' | Done | API |
| A | Make a test suite in postman for the route '/api/passwords/change' | Done | API |
| A | As a API i must have a route /api/miner/:MAC_Address/:customer[GET], that returns detailed information about a given miner | Done | API |
| A | As a API i must have the route /api/map/:customer[POST] that will insert location into Miner table | Done | API |
| A | As a API i must have the route /api/subnets/:customer[GET], that returns the latest information about all subnets for a given customer | Done | API |
| A | Add User_logs and Miner_Logs tables to the loadTables.sql script | Done | API |
| A | Make a test suite in postman for the route '/api/databases | Done | API |
| A | Make a test suite in postman for the route '/api/subnets/:customer/:subnet/'[post] | Done | API |
| A | Log all Authentication errors that occur into a Log table | Done | API |
| A | As a API i must have the route 'api/log/miner/:customer'[GET], that returns the miner logs for a given customer | Done | API |

| | | | |
|---|---|---|---|
| A | On insertion log into the Miner_Logs if the miner has a critical value | Done | API |
| A | Create a HIgh level component diagram for the API | Done | API |
| A | Change the data that each route returns, use the model that we discussed as a group | Done | API |
| A | Make a test suite in postman for the route '/api/miners/:subnet/:customer' | Done | API |
| A | Model the return data from each route in the API | Done | API |
| A | Add an authentication to all routes | Done | API |
| A | As a API i must have a route /api/miners/:customer[POST] that inserts given data into the miners tables | Done | API |
| A | Log all error that occur in the API in the error table | Done | API |
| A | Implement a data validation for all routes in the api | Done | API |
| A | Route '/api/miners/pod/:id/:customer', that returns all miners wihtin a given pod | Done | API |
| A | as a api i must have a route 'api/houses/:customer[GET], that gets all houses for the given customer | Done | API |
| A | as a api i must have a route 'api/pods:/houseID/:customer'[GET], that get all pods for a given house | Done | API |
| A | as a api i must have a route 'api/house/:customer'[POST], that adds a house entry to a given customer | Done | API |
| A | as a api i must have a route 'api/pod/:customer'[POST], that adds a pod entry to a given customer | Done | API |
| A | Create a test suite for, getPods, getHouses, insertPod, insertHouse, getMinersByPod | Done | API |
| A | Change the API so all routes work remotely | Done | API |
| A | as a API i must have a route /api/pool/:id/:customer[DELETE], that deletes the pool with the given id | Done | API |
| A | as a API i must have a route /api/house/:id/:customer[DELETE], that deletes the house with the given id | Done | API |
| A | as a API i must have a route /api/pod/:id/:customer[DELETE], that deletes the pod with the given id | Done | API |
| A | Add online/offline miners to the return body for all customers | Done | API |
| A | Add privilege information in either the database route or the get subnets route | Done | API |
| A | Add hashrate and online/offline information to all pods in the get pods route | Done | API |
| A | Add a get Miner_Log by mac route to the API | Done | API |
| A | Add 0 hashrate if mintue interval is not availabe in the get hashrate route for all subnets | Done | API |
| A | when user changes password update the password in the User_Table in the Admin database | Done | API |
| A | Go over AP automated tests on server for the past few days and debugg if en error occured | Done | API |
| A | Fix the data we get from the getHash by min/hour/day route | Done | API |
| A | Setup the newest version of API, DB on the mariaDB at M15 | Done | API |
| A | As a administrator i need to create a script that fills the database with fake data | Done | Database |

| | | | |
|---|---|---|---|
| A | As a administrator i need to create a script that populates the database | Done | Database |
| A | Create a error log tables that resides within the a admin database that logs all errors that occur, (Auth_Log, AccessDenied_Log, API_Log) | Done | Database |
| A | Add changes made to the database to an ER diagram | Done | Database |
| A | Setup API / db / scanner up at Mjölnir | Done | Database |
| A | Sequence diagram for scanner | Done | Design |
| A | Create a high level component diagram | Done | Design |
| A | Create an ER diagram | Done | Design |
| A | Sequence diagram for API for get routes | Done | Design |
| A | Wireframe mockup for website(aka design the UI) | Done | Design |
| A | Discuss how we want to test | Done | Design |
| A | Do a mockup for the change password view and the User_Log view | Done | Design |
| A | Prepare for our first status meeting | Done | Meetings |
| A | Project Preperation | Done | Preparation |
| A | Create a peer presentation of the project | Done | Preparation |
| A | Decide story points | Done | Preparation |
| A | Set up a GIT Repo | Done | Preparation |
| A | Set up Time overview | Done | Preparation |
| A | Open connection between the server and the network in our workspace | Done | Preparation |
| A | Setup the server running the DB and the API | Done | Preparation |
| A | Dev environment and database set up troubleshooting | Done | Preparation |
| A | Prepare for first peer presentation | Done | Presentation s |
| A | Presentation for third status meeting | Done | Presentation s |
| A | Create an procedural agreement | Done | Report |
| A | Create a risk assessment | Done | Report |
| A | Create an progress report | Done | Report |
| A | Create an design report | Done | Report |
| A | Final Report | Done | Report |
| A | As a scanner i have to be able to scan the subnet for Ip/Mac | Done | Scanner |
| A | As a scanner i have to be able to calculate total hash/power for the whole subnet. | Done | Scanner |
| A | Establish a TCP connection to the miners from our workspace | Done | Scanner |
| A | As a scanner I have to be able to look up cg miner version in a lookup table. | Done | Scanner |
| A | As a scanner I have to be able to send version command to each ID/Miner and use the response to get the next command and the fetch state. | Done | Scanner |

|  |  |  |  |
|---|---|---|---|
| A | As a scanner I have to be able to iterate through the IP list and establish a TCP connection. | Done | Scanner |
| A | Unit test the scanner | Done | Scanner |
| A | As a scanner i have to be able to execute a command and fetch the information. | Done | Scanner |
| A | Disscuss how we want to implement the application that will send commands to miners | Done | Scanner |
| A | Set up raspberry pi and test scanner on the pi connected to subnet | Done | Scanner |
| A | Research the way manufacturers are sending commands to miners by sniffing traffic | Done | Scanner |
| A | As a scanner i have to be able to send the info/mac/ip to the API, sleep for 5min. Repeat the process | Done | Scanner |
| A | Figure out how to pull only the scanner into raspberry Pi | Done | Scanner |
| A | Optimize and polish out redundant code in scanner/command server | Done | Scanner |
| A | Fix heat values in scanner where they are too long and take max value | Done | Scanner |
| A | Split up scanner return data efficiently so API can handle it | Done | Scanner |
| A | Scanner debugging | Done | Scanner |
| A | Fix the data units for the speed properties, and remotely connection to the command server from the API | Done | Scanner |
| A | Create template for the website. | Done | Website |
| A | Connect Website to API | Done | Website |
| A | Make a route between all the subsites | Done | Website |
| A | Fully implement Redux into React. | Done | Website |
| A | As a user i want to see all subnets/pods a client owns(client site). | Done | Website |
| A | Update the database, add House and pod tables, update fields in Miner and Miner_Info | Done | Website |
| A | As a user i want to be able to see a list of clients i service. | Done | Website |
| A | As a user i want to be able to see a list of miners within a certain subnet | Done | Website |
| A | Setup the navigation bars | Done | Website |
| A | as a user i want to be able to change my password | Done | Website |
| A | Create Miner_Log view for the website | Done | Website |
| A | As a user I have to be able to log into the website. | Done | Website |
| A | As a user i want to be able to select a single miner and get all the information about that miner. | Done | Website |
| A | As a user i want to be able to map the location of miner/miners under the map sub route | Done | Website |
| A | In the map configuration change the house input to a dropdown where we pick from the house table | Done | Website |
| A | In the map configuration change the pod input to a dropdown where we pick pods from the house that is picked | Done | Website |
| A | in client view a list of houses that reside within each subnet | Done | Website |
| A | Change the store each reducer has error ok, msg, data properties | Done | Website |

| | | | |
|---|---|---|---|
| A | Add a House view that displays all the pods that reside within the house | Done | Website |
| A | Clean code and Add proptypes to all functional components | Done | Website |
| A | Add a Pod view that displays a heatmap/list of all miners within the pod | Done | Website |
| A | Add a configuration view where customers can add a house and add pod to each house | Done | Website |
| A | as a user i must be able to delete a entry of a pool | Done | Website |
| A | as a user i must be able to delete a entry of a pod | Done | Website |
| A | as a user i must be able to delete a entry of a house | Done | Website |
| A | Add a privilege check in the configure route | Done | Website |
| A | Add measurment units to all numbers on the website | Done | Website |
| A | Display a Miner_Log for a particular miner in the miner details view | Done | Website |
| A | Research login forms, sidebars, custom tables for hangout meetings | Done | Website |
| A | Add WorkerName, IP_Address, time elapsed and offline/online to the miner list table | Done | Website |
| A | Add a sort by proerty functionality to the Tables | Done | Website |
| A | add a slider to the MinerList that will utilize the paging functionality | Done | Website |
| A | Create a hashrate graph our of the Subnet_Info data | Done | Website |
| A | Get scanner to run on raspberry pi again | Done | Website |
| A | add sprint burndown chart to sprint 1 and 2, and fix the burndown for sprint 6 | Done | Website |
| A | Re-style miner details view | Done | Website |
| A | Fix the subnet/house/ connection after what was discussed with Product Owner | Done | Website |
| A | Implement some kind of a refresh mechanishm on current page, and fix the redirect to the dashboard on page refresh | Done | Website |
| A | Implement client view hashrate graph using D3 | Done | Website |
| A | update the ER diagram | Done | Website |
| A | Finish design report | Done | Website |
| A | Finish progress report for sprint 1-7 | Done | Website |
| A | style the heatmap | Done | Website |
| A | Style the table in minerList | Done | Website |
| A | add extra authentication to the LDAP | Done | Website |
| A | Rewrite code for the heatmap and comment it. | Done | Website |
| A | fixing minor bugs | In progress | Website |
| B | as a API i must have a route to /api/passwords/change/[POST], that changes the password of the current user | Done | API |
| B | As a API i must have the route /api/hash/:customer[GET], that returns the total hash since the user started using the system | Done | API |

| | | | |
|---|---|---|---|
| B | Make a test suite in postman for the route '/api/map/:customer' | Done | API |
| B | Automate Postman tests | Done | API |
| B | Add a get Page/index, to get Miners in the API | Done | API |
| B | Add a get page/index, to get User_Logs in the API | Done | API |
| B | Add a get page/index, to get Miner_Logs in the API | Done | API |
| B | order paging | Done | API |
| B | Add a check on map if the house is connected to the subnet that the miner is being mapped to | Done | API |
| B | New more detailed mockup of the website. | Done | Design |
| B | Discuss as a team the theme/images we want to use for styling the website | Done | Design |
| B | Prepare for second status meeting | Done | Meetings |
| B | Prepare for third status meeting | Done | Meetings |
| B | Add error logging to scanner | Done | Scanner |
| B | Automate scanner tests | Done | Scanner |
| B | error checka command server (þegar version er náð og þegar commands eru sent) | Done | Scanner |
| B | Research work for disable hashing on s9 | Done | Scanner |
| B | Setup addtional scanner | Done | Scanner |
| B | Finish commenting both scanner and command server | Done | Scanner |
| B | skoða nonce og aðra fields í digest header og gera það dynamic | Done | Scanner |
| B | Create a messagepanel that handles all success and error messages | Done | Website |
| B | Implement the message panel in all views | Done | Website |
| B | As a user i want to be able to see a heatmap of each subnet/pod(subnet/pod site). | Done | Website |
| B | style subnet view | Done | Website |
| B | style change password view | Done | Website |
| B | Create a paging functionality from template | Done | Website |
| B | style house view | Done | Website |
| B | style map miners view | Done | Website |
| B | style pod view | Done | Website |
| B | implement a mapping functionality in the minerView (as the last slide in the carousel) | Done | Website |
| B | Create a sidebar from the first template | Done | Website |
| B | style configure view | Done | Website |
| B | style dashboard view | Done | Website |
| B | implement the miner view as a carousel | Done | Website |
| B | style user log view | Done | Website |
| B | style client view | Done | Website |
| B | Style the minerLog view | Done | Website |
| B | style log in view | Done | Website |

| | | | |
|---|---|---|---|
| B | style miner details view | Done | Website |
| B | implement sort for miner/user logs tables | Done | Website |
| B | Implement some kind of navigation tree to navigate back to visited sites | Done | Website |
| B | Change how we display the subnets and houses in client view | Done | Website |
| B | Add online/offline pie chart in the dashboard | Done | Website |
| B | Change the theme of the topbar/sidebar, follow the theme at https://advaniadc.com/data-center-solutions | Done | Website |
| B | Style the website | Done | Website |
| B | Add online/offline pie chart in the house View | Done | Website |
| B | seperate the miners in minerStore for pod, one for the table and another one for the heatmap | Done | Website |
| B | Clean out all console logs | Done | Website |
| C | As a API i must have a route /api/pools/:customer[GET], that returns all pools for a given customer | Done | API |
| C | As a API i must have a route /api/pool/:id:customer[PUT], that edits the pools info for the given pool ID | Done | API |
| C | As a API i must have a route /api/pools/:customer[POST] that inserts given information into the customers pool table | Done | API |
| C | As a API i must have the route 'api/logs/user/:customer'[GET], that returns the user logs for a given customer | Done | API |
| C | As i API i must have a route '/api/logs/user/:time/:customer', that returns a list for MAC addresses that are linked to given time | Done | API |
| C | Make a test suite in postman for the route 'api/logs/user/:customer' | Done | API |
| C | when user executes a command log it into the User_Log table | Done | API |
| C | Create a documentation/readme for the API | Done | API |
| C | As a API i must have a route /api/commands/:command/:customer[POST], that will send a command to a miner | Done | API |
| C | Add measument units to getHash route | Done | API |
| C | change the getHash route in the api to getHashByDay, getHashByHour, getHashBy5min | Done | API |
| C | Add a delete favorite miner(s) route to the API | Done | API |
| C | Remove favourite pod/house feature to API | Done | API |
| C | Add getFavorite items route to the API | Done | API |
| C | Add get favorite miners route to the API | Done | API |
| C | Add a insert favorite miner(s) route to the API | Done | API |
| C | Add microsoft auth to the API/DB | Done | API |
| C | Add favourite pod(s)/house(s) feature to API | Done | API |
| C | Add a paging option for the user in slider | Done | API |
| C | change login and change password to get routes, using auth header instead of body/POST route | Done | API |
| C | Split the API into more controllers/services | Done | API |

| | | | |
|---|---|---|---|
| C | add a prefix House / Pod / to all miner locations on the website | Done | API |
| C | make the time interval between data points in getTotalHash dynamic | Done | API |
| C | group the send command miners by subnet id and send to each subnet the command | Done | API |
| C | NPM vesen og setup á local database | Done | Preparation |
| C | As a server i must send back a result message to the client | Done | Scanner |
| C | As a client i must be able to send a list of ips, command pair to the server | Done | Scanner |
| C | create a template for the server and client, that will be used to send commands to miners | Done | Scanner |
| C | as a server i must iterate though a given list of ips and send to them requested command | Done | Scanner |
| C | Figure out how to work with antminer restart response in command server to update success/failure | Done | Scanner |
| C | As a server i must read from a configuration file to get auth information for the miners | Done | Scanner |
| C | As a scanner i want to be able to run on startup, using a configuration file for subnet size, subnet_id, and all related variables | Done | Scanner |
| C | modify command server for whatsminer | Done | Scanner |
| C | As a user i want to be able to see list of configs that a customer has under the pools sub route | Done | Website |
| C | As a user with add privileges i want to be able to add config information to a list of configs | Done | Website |
| C | As a user i want to be able to send commands to a specific miner (f.e. restart) | Done | Website |
| C | As a user i want to be able to config/restart a range of miners | Done | Website |
| C | In UserLog display all miners onclick, that are assosiated with a timestamp | Done | Website |
| C | Add extra subroute in the side navigation bar for the user_log that only admins see | Done | Website |
| C | implement a redux store for the pagingSize | Done | Website |
| C | Add a ldap office 365 to the website | Done | Website |
| C | Fix how different viewers see configure view | Done | Website |
| C | Style the navigation tree | Done | Website |
| C | Add a redirection if there is only one client available for the user | Done | Website |
| C | Add sidebar to dashboard view only with favorite items | Done | Website |
| C | implement the configure view as a tab view | Done | Website |
| C | Add favourite pod/house feature | Done | Website |
| C | research tab view for miner details view | Done | Website |
| C | Add a functionality to order by favorites | Done | Website |
| C | add extra information in the getPods route, like zero hash and overheat | Done | Website |
| C | Add some styles to the pool information table | Done | Website |

| | | | |
|---|---|---|---|
| C | Reorder the house and subnets side by side with row wrap in client view | Done | Website |
| C | Add a favorite item in the sidebar that's a parent element of house and pod | Done | Website |
| C | Add missing Asiic and type field to miner table | Done | Website |
| C | fix the logout on change password, and set a error message if the passoword dose not meet requirements | Done | Website |
| C | add maximize and minimize functionality to the podListLitems in houseView | Done | Website |
| C | Style dashboard components with a graph and list of houses/pods | Done | Website |
| C | Take out borders from all buttons when they are selected | Done | Website |
| C | Have the inputs in map view scrollable like the Heatmap | Done | Website |
| C | Change the get Hash by min, hour, day from button to tabs | Done | Website |
| c | Add the pie chart from house view to subnets and houses in the client view | Done | Website |
| c | Setup subnets, houses, and map single miner as a panel similar to the panel in the dashboard view | Done | Website |
| c | fix errors and warnings | Done | Website |
| C | change the clientStore so it stores houses/pods for the tables in the dashview | Done | Website |
| C | implement a minimize/maxize for clients in dashboard view | Done | Website |
| C | setup singlemapView, houseConfigure, podConfigure, connectionConfigure, poolConfigure to the card theme like the rest of the side | Done | Website |
| | Code commenting | Done | Website |

## Risk assessment

Below is a list of all Risks from our risk assessment that affected the team during the project's duration, how we defined them and how they were dealt with.

| Risk | Description | Rating | Odds | Effect | Risk Value | Accountable |
|---|---|---|---|---|---|---|
| 3 | Sickness/Personal Problems | 1 | 3 | More workload on rest of the team | 3 | Rest of the team |
| 24.1.2020 - Birkir was sick when we met with our instructor – this had little effect since it was only one meeting. | | | | | | |
| 5 | Missing hardware or software | 4 | 1 | Unable to run affected parts of the system | 4 | Kristmann |
| 6.3.2020 - Got a server running the API and MariaDB, odds down to 1 from 2 | | | | | | |
| 6 | Hardware failure of work computers | 3 | 1 | Team member is without a computer and needs to work from his | 3 | Computer owner |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
|  |  |  |  | home computer or borrow a computer |  |  |
| 26.3.2020 - The fan on Birkir's laptop malfunctioned and had to be in the shop for maintenance for 2-4 weeks, but he borrowed a computer from a friend as a substitute in the meanwhile. |  |  |  |  |  |  |
| 7 | Insufficient data from ping | 3 | 1 | Wrong/non data will be pulled from the miners | 3 | Kristmann |
| 13.1.2020 - we have data from December preparation that does ping on most types of miners. Therefore, the odds were changed to 1 from 3<br>18.2.2020 - we were supposed to get an ethernet connection to our workspace to a miner subnet, but it was not ready, so we weren't able to fetch data. Odds went up to 2<br>25.2.2020 - we got the ethernet connection but were refused by the hosts for connection. Odds went up to 3<br>27.2.2020 - At least 3 miners vary in returning hash from all asic's, might have to find out later how to get that information and set up edge cases. Odds don't go up since it is such a small number of miners having trouble.<br>2.2.2020 - we solved our issues for now, set up edge cases so the scanner can run uninterrupted and returns all available results for miners. Odds went down to 1. |  |  |  |  |  |  |
| 10 | Remote connection to DB issues | 3 | 1 | Halted development due to inability to fetch data. | 3 | Birkir |
| 17.2.2020 - managed to connect remotely to mysql server that was on local network. Odds down to 1 from 2<br>6.3.2020 - managed to connect to the centos server that running the API and MariaDB. |  |  |  |  |  |  |
| 14 | Issues hosting the API | 4 | 0 | No communication between DB, scanner and website | 0 | Birkir |
| 6.3.2020 - finished setting up the centos server that is hosting the api, odds down to 0 from 1 |  |  |  |  |  |  |
| 19 | Problems with authentication for sending commands to miners | 2 | 1 | Hinders or delays us from completing all tasks associated with commands | 2 | Kristmann |
| 10.3.2020 - authentication problem came up and we're looking for solutions. Odds changed from 2 to 5<br>12.3.2020 - we found an external API on the miners to send commands to. Odds changed from 5 to 1 |  |  |  |  |  |  |

## Terms

| Term | Description |
|---|---|
| Client | A Client is a user on the website that has access to one or more customers. Employees of Advania datacenters are thought of as clients |
| Customer | A customer is a user that has his own database, a customer only has access to his own database. |
| Miner(s)/worker(s) | Machine(s) that calculates transactions that occur on the bitcoin network at high speed. |
| Pod | A pod is a collection of miners. Usually one house is divided into 4 pods. |
| Hashrate | Hash rate is the measurement of a miner's performance. In other words, it is the hash function's output, or it is the speed at which a miner solves the Bitcoin code. ... Hash/s is also measured in J/Ghash (Joules per 1 Billion hashes). Hash per second represents SHA-256 algorithms that are used per second, known as hash rate |
| Farm | Farm is the collection of all miners that a customer has. |
| Block | A block is a collection of all bitcoin transactions that have been performed in the last 10 minutes. |
| Pool | Bitcoin mining pools are a way for Bitcoin miners to pool their resources together and share their hashing power while splitting the reward equally according to the number of shares they contributed to solving a block. |