



**B.Sc. final project – Computer Science**

# Transmo

Transforming graphs to data

**May, 2022**

**Name of student:** Finnbogi Jakobsson

**Kennitala:** 130698-2399

**Name of student:** Gísli Þór Gunnarsson

**Kennitala:** 101298 - 2219

**Name of student:** Hákon Hákonarson

**Kennitala:** 021098-2489

**Supervisor:** Sigurjón Ingi Garðarsson

# Table of contents

1	Introduction.....	1
1.1	Background.....	1
1.2	Project description .....	1
2	System design .....	1
2.1	System flow .....	1
2.2	Tech stack .....	2
2.2.1	Front-end.....	2
2.2.2	Back-end web service .....	2
2.2.3	Back-end image processing .....	3
2.3	System overview .....	3
2.4	Image processing .....	4
2.4.1	Three problems .....	5
2.4.2	Detecting slices in a pie .....	6
2.4.3	Finding text in an image .....	8
2.4.4	Combining text and pie slices .....	9
2.5	API: application programming interface .....	9
2.6	Wireframes of the frontend.....	10
2.7	Testing.....	11
2.7.1	Unit testing.....	11
2.7.2	Accuracy testing.....	12
3	Risk analysis .....	14
4	Product backlog .....	16
5	Progress report .....	19
5.1	Project burndown .....	19
5.2	Sprint overview .....	20
5.2.1	Sprint 0.....	20

5.2.2	Sprint 1 .....	20
5.2.3	Sprint 2.....	20
5.2.4	Sprint 3.....	20
5.2.5	Sprint 4.....	21
5.2.6	Sprint 5.....	21
5.2.7	Sprint 6.....	21
5.2.8	Sprint 7.....	21
6	Known bugs .....	21
6.1	Text algorithm.....	21
6.2	Dark coloured background.....	21
6.3	Size check of uploaded file .....	21
6.4	Graphs that are not made in Excel .....	22
6.5	GET transmo-api.grid.is/finn/bogi.....	22
6.6	Removal of colour icons .....	22
7	Conclusion .....	22
7.1	Final product .....	22
7.2	Retrospective.....	22
7.2.1	What went well .....	23
7.2.2	What could have gone better.....	23
7.2.3	Future development .....	23
8	Appendix.....	24
8.1	Sprints .....	24
8.1.1	Sprint 0.....	24
8.1.2	Sprint 1.....	25
8.1.3	Sprint 2.....	28
8.1.4	Sprint 3.....	31
8.1.5	Sprint 4.....	33

8.1.6	Sprint 5.....	35
8.1.7	Sprint 6.....	38
8.1.8	Sprint 7.....	39
8.2	Test images .....	42
8.3	Test data .....	46
8.4	Work distribution by work type .....	49

## Table of figures

Figure 1: Flow chart.....	2
Figure 2: Prototype of the system overview .....	4
Figure 3: Original image .....	5
Figure 4: Everything masked out except the pie.....	6
Figure 5: Pie masked out.....	7
Figure 6: Colour icons removed .....	7
Figure 7: Image thresholded .....	8
Figure 8: API.....	10
Figure 9: Frontend wireframe 1 .....	1
Figure 10: Final look of the frontend .....	1
Figure 11: Project burndown .....	1
Figure 12: Work distribution .....	1
Figure 13: Sprint 1 burndown .....	1
Figure 14: Sprint 2 burndown .....	2
Figure 15: Sprint 3 burndown .....	2
Figure 16: Sprint 4 burndown .....	2
Figure 17: Sprint 5 burndown .....	3
Figure 18: Sprint 6 burndown .....	3
Figure 19: Sprint 7 burndown .....	4
Figure 20: 12_entries .....	5
Figure 21: fiveshadesofRED_20values_appart.....	6
Figure 22: bogatest.....	8
Figure 23: fiveshadesofRED_10values_appart.....	9

Figure 24: fiveshadesofgray.....	9
Figure 25: rokk_stig.....	10
Figure 26: 8_entries .....	11
Figure 27: 11_entries .....	11
Figure 28: 12_entries .....	12
Figure 29: Finnbogi, time spent by work type.....	14
Figure 30: Gísli, time spent by work type.....	16
Figure 31: Hákon, time spent by work type.....	19

## Table of tables

Table 1: Unit test coverage .....	12
Table 2: Risk analysis .....	14
Table 3: Product backlog .....	16
Table 4: Total time spent .....	19
Table 5: Sprint 0 backlog .....	24
Table 6: Sprint 0 time spent .....	24
Table 7: Sprint 1 backlog .....	25
Table 8: Sprint 1 time spent .....	27
Table 9: Sprint 2 backlog .....	28
Table 10: Sprint 2 time spent .....	31
Table 11: Sprint 3 backlog .....	31
Table 12: Sprint 3 time spent .....	33
Table 13: Sprint 4 backlog .....	34
Table 14: Sprint 4 time spent .....	35
Table 15: Sprint 5 backlog .....	36
Table 16: Sprint 5 time spent .....	37
Table 17: Sprint 6 backlog .....	38
Table 18: Sprint 6 time spent .....	39
Table 19: Sprint 7 backlog .....	40
Table 20: Sprint 7 time spent .....	41
Table 21: 12_entries.....	46
Table 22: fiveshadesofRED_20values_appart.....	46
Table 23: bogatest.....	47

Table 24: fiveshadesofRED_10values_appart .....	47
Table 25: fiveshadesofgray .....	47
Table 26: rokk_stig .....	47
Table 27: 8_entries.....	47
Table 28: 11_entries.....	48
Table 29: 10_entries.....	48

# **1 Introduction**

## **1.1 Background**

The idea for this project originally came from the web development company Grid. Grid was founded in 2018 by "a group of web software veterans and data enthusiasts" as they describe themselves and their goal is to reinvent the way organizations and their people work with data and numbers. They are doing this by developing their product which is an innovative no-code web tool which can help people optimize their data by integrating with their spreadsheet software and allows them to work quickly and strategically in a familiar spreadsheet environment. Their product can help produce beautiful web reports and interactive scenarios without first having to learn a complex new program.

## **1.2 Project description**

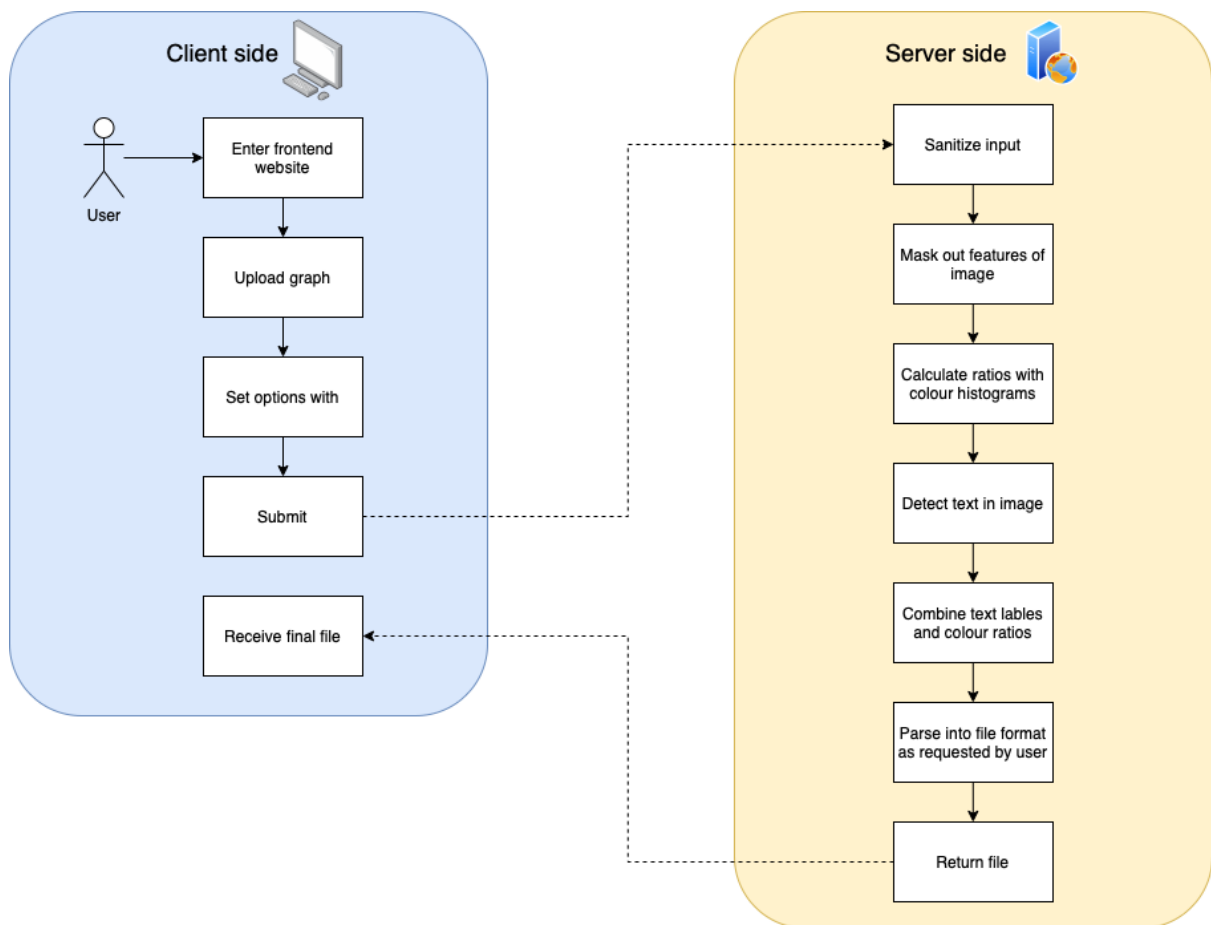
On the internet there are many reports and articles which use graphs to draw attention to certain findings they want to emphasize. While the human eye can easily read these graphs and understand the data they are intended to show, it is not as easy for a computer. So, when the need arises to do further research with the data and only the graphs are available it could get quite tedious and costly to input it all manually, that is where this project could be of use. The main goal of this project is to create software prototype which can read these graphs and automatically write the data behind them to a spreadsheet. This could save both time and resources for anyone doing research based on graphs when it is hard or impossible to get the original data.

# **2 System design**

## **2.1 System flow**

A user will upload an image of a graph on a website which will be sent to another server through an API. That server then starts a process with a text reader and colour histogram that will analyse the image and extract all the data that was used to create the graph. Put that data into a .xlsx or .csv file which is then returned to the user.

Figure 1: Flow chart



## 2.2 Tech stack

The system consists of the following items.

### 2.2.1 Front-end

- **Vue**

A frontend framework for developing single page applications. A good tool to create functioning and good-looking website without coding everything from scratch.

- **JavaScript**

Programming language for frontend development.

### 2.2.2 Back-end web service

- **Flask**

A python web service framework. Used to create python written backends for websites and APIs.



- **Python**

One of the most popular high-level general-purpose programming languages.

- **AWS – Elastic Bean Stalk**

A web service hosting provider.

### 2.2.3 *Back-end image processing*

- **Python**

One of the most popular high-level general-purpose programming languages.

- **OpenCV**

A library used in python for various of different image processing tasks.

- **Pytesseract**

A library that uses machine learning to detect and read text.

- **Pandas**

A library that is used in various of data science projects to encapsulate data into an efficient data frame.

### 2.2.4 Version control and other supporting software

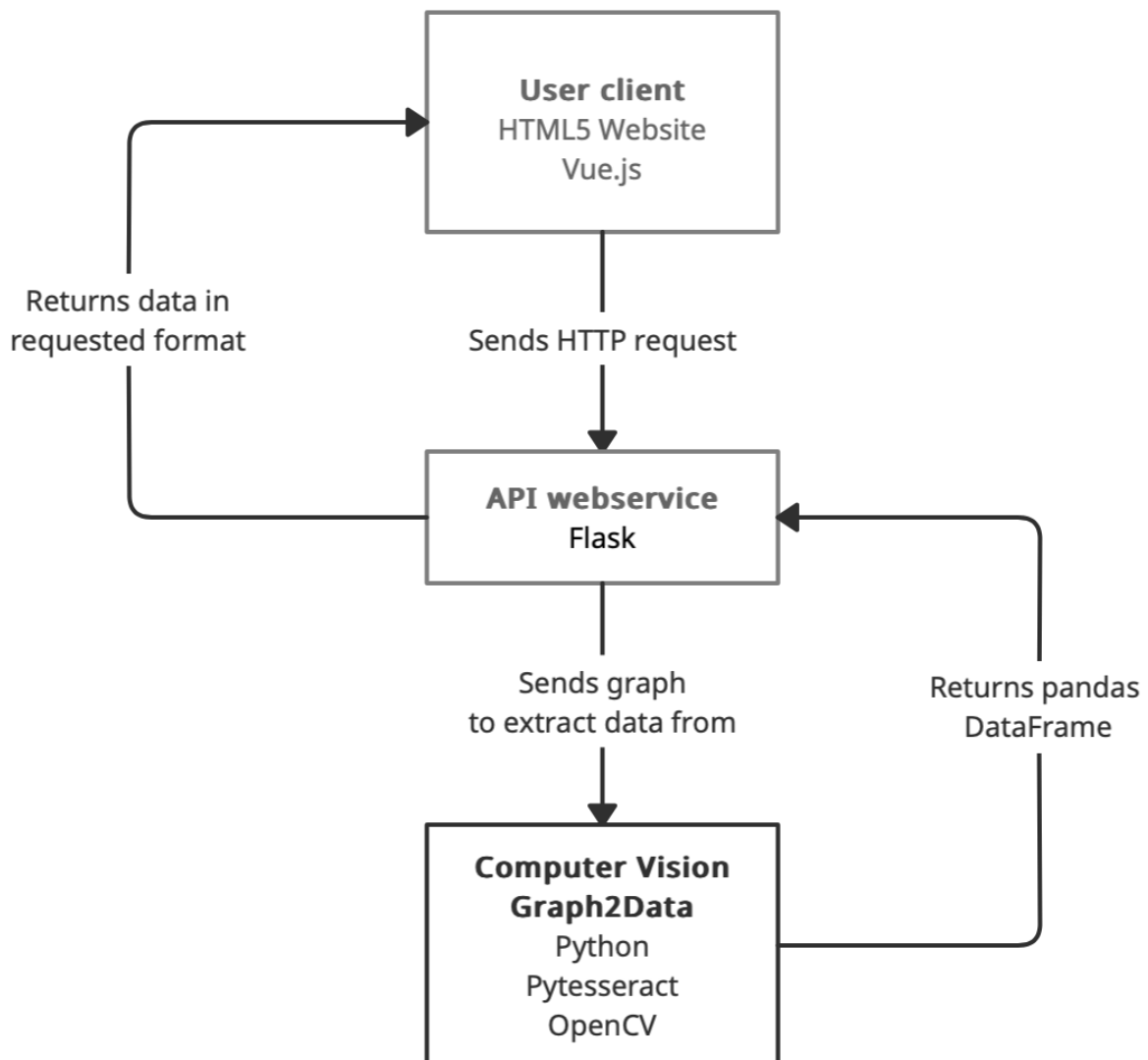
- **GitHub**

The most popular version control software for programming. A useful tool when collaborating on a project so different collaborators don't overwrite each other's work.

## 2.3 System overview

The system is composed of three main elements. The front-end which is a website for users to interact with the system. A back-end comprised of two elements, one is a web server API that accepts the requests from the website and the other is a logic-layer element which manages the graph algorithm. This way the three systems are independent and can be easily worked on separately. For example, it would be easy to swap the front end out or add another website. Same goes for the computer vision element, if a better computer vision algorithm is developed it could be changed out without affecting neither the API nor the front end. Similar goes with the API as long it accepts the same API commands.

Figure 2: Prototype of the system overview

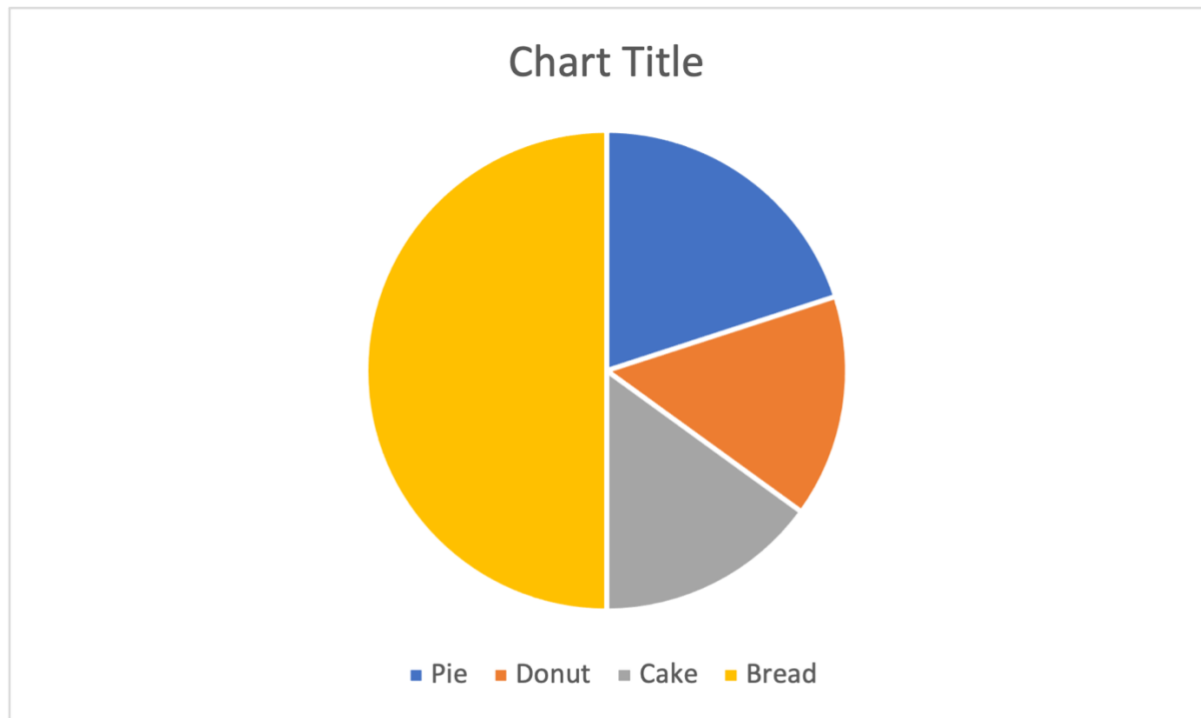


## 2.4 Image processing

The implementation ended up being quite different than what was set out to be based on the original project description. The original project description said to make a program that would take most graph types and extract the data behind that graph into a table. The reverse of creating a graph in a sheet program like excel. The team consulted with a teacher in Reykjavík University that did his Ph.D. in computer vision on how to approach this project due to lack of knowledge in creating a system of this complexity. The outcome of that meeting was that the project was bigger and more complicated than anticipated and the team needed to figure a way to scale down the problem to something simpler. With that information the team brainstormed and produced a smaller version which only detects pie

charts to begin with and if that is completed quickly then more graph types could be added. Pie charts were chosen because it is possible to detect the size of each slice without training a machine learning model.

*Figure 3: Original image*



### ***2.4.1 Three problems***

Detecting a pie chart involves three main subproblems. First is to detect the size of each slice in the pie chart. Second is to detect the label positions and the actual text that makes up the label. Third is to connect the labels to the slices so the results returned by the system have any meaning. Each of these problems in and of themselves are difficult for a team of students but not impossible.

*Figure 4: Everything masked out except the pie*

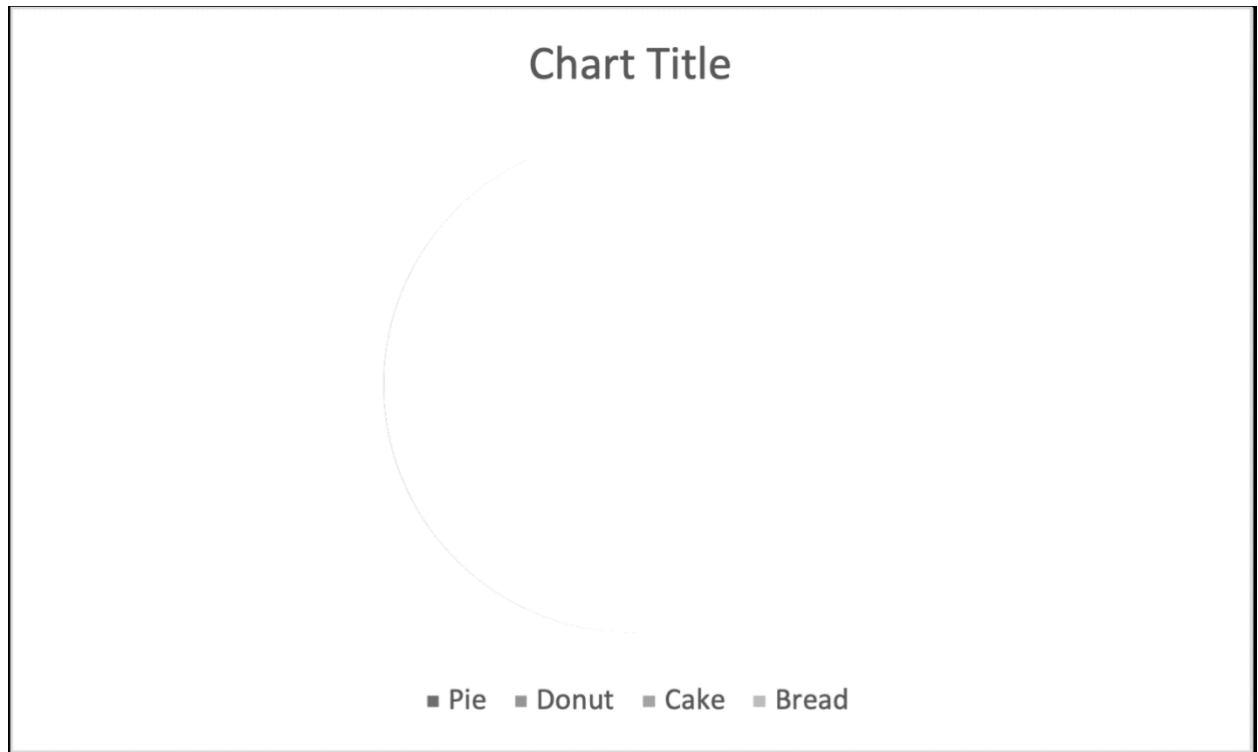


#### ***2.4.2 Detecting slices in a pie***

Detecting the pie and its slices was the first obstacle. Using a colour histogram, the pixel count of a given colour value can be more easily organized or binned. For that, a library called OpenCV came in very handy. It is often used in machine learning applications, but it has good tools for image analysis and manipulation. For this project it was used to bin similar colours together with colour histogram function and to find a big circle in the image that is most likely the pie and mask out everything else in the image, so the colour histogram is more effective. The team had to figure out what bin size to use, too big and few and the program might end up binning similar but separate colours together. Having too many bins takes up more memory and makes the time complexity of going through them bigger and in some cases a single slice in a pie could have one colour to the human eye, but the exact colour value could vary by one or two values which would result in one slice ending up in multiple bins, which is not ideal. Ideal number of bins and the size of each bin is to be a power of two since computers are efficient if it can work in powers of two. After some excel calculations and speculation the team ended up having the bin size 8 and number of bins 32. This should catch all pixels of same colour with slight variable value, but if a colour sits right on the value where a bin ends and a new bin begins the program deals with that by checking neighbouring bins to the bin that has the most pixels. Having the bins larger and fewer

without neighbour checks is not as robust since there is still a chance a colour lands on the intersection where two bins meet and then the pixel count could split into two bins. That is why the team decided on slightly smaller bins with neighbour checks.

*Figure 5: Pie masked out*



*Figure 6: Colour icons removed*

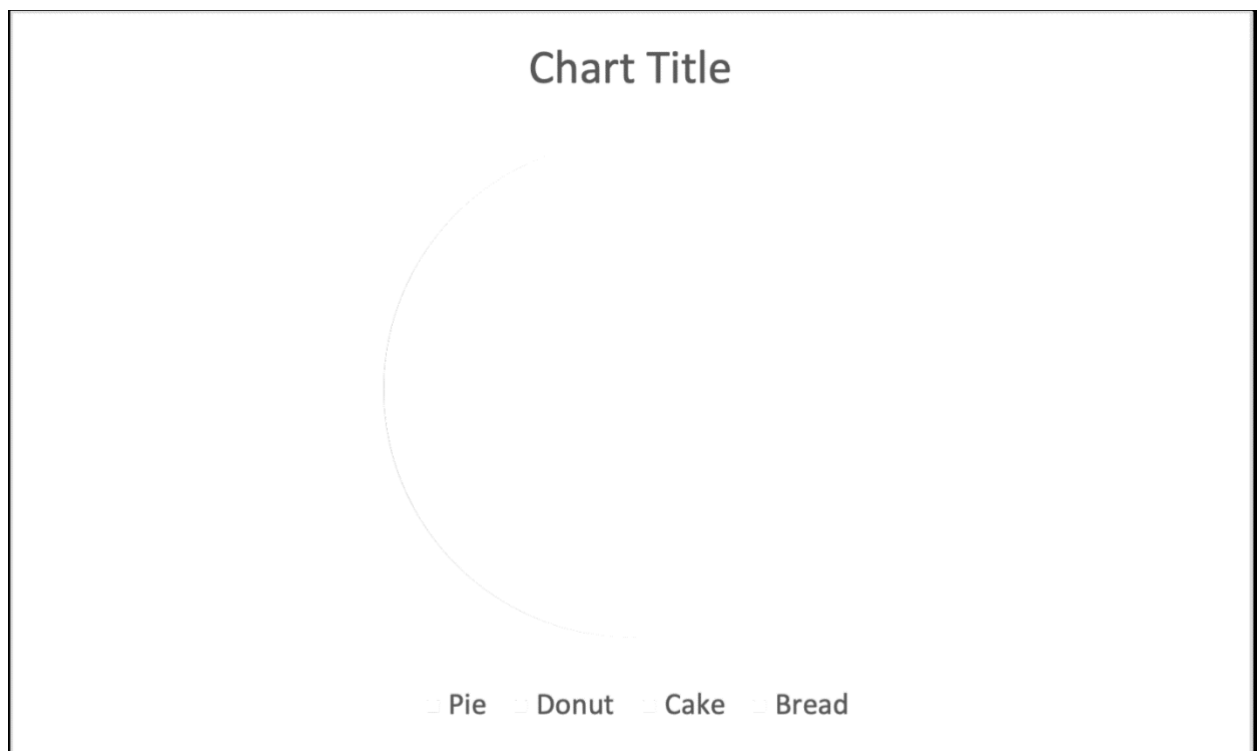
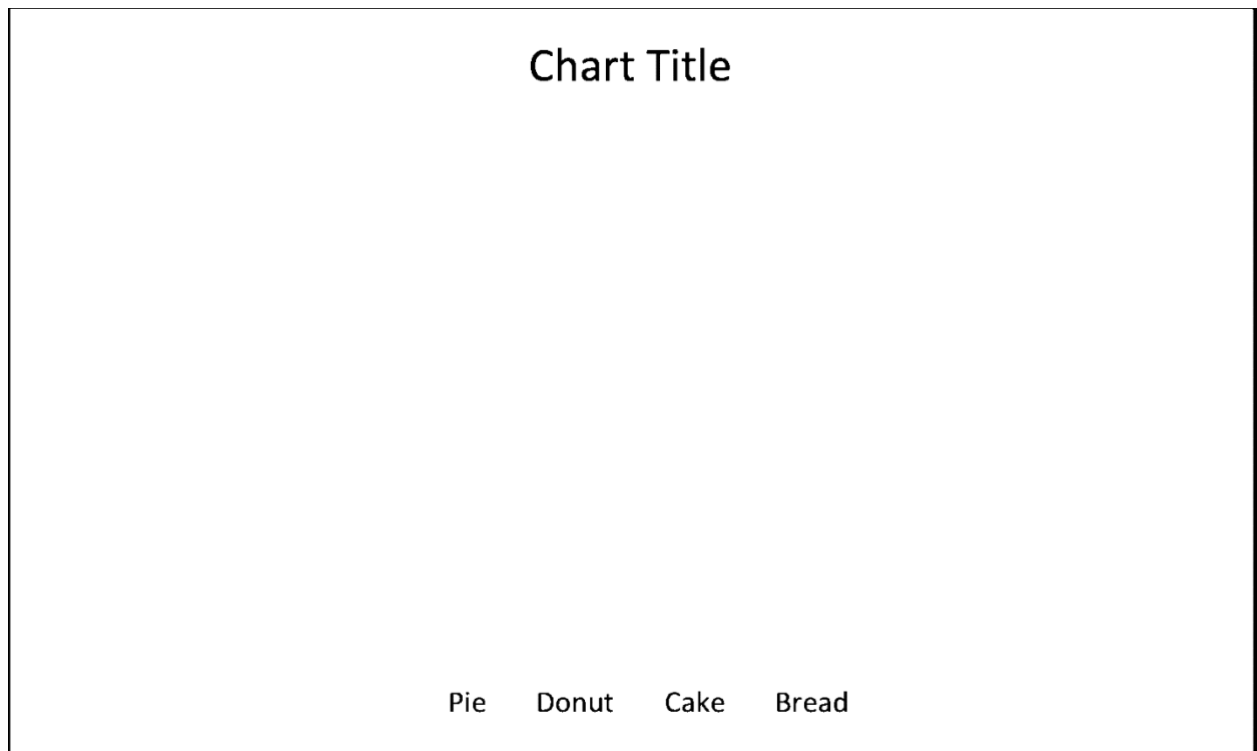


Figure 7: Image thresholded



### 2.4.3 Finding text in an image

Finding and reading the text was the most difficult obstacle of them all. For this problem a library called Tesseract OCR was used instead of implementing text detection from scratch. Tesseract OCR is an engine owned by Google and originally developed by Hewlett-Packard; it is currently free to use under the Apache license. Before this is used a few image processing steps are taken with OpenCV to make the text detection easier. Some of the steps taken are masking out the pie in the image so there are less things to confuse the engine, remove noise, remove the rectangular icons and then threshold the image. Thresholding makes very light pixels white and darker pixels are made black to have the most contrast for the character detection. The image is sent through the Tesseract OCR engine, and it returns all text that it finds in the image, also positions of each text, certainty of what it detected and other information. This information is then processed to figure out what text were the actual labels and what could be discarded. To get to this point a lot of work and research was needed to understand the engine and what parameters would give the best results and figuring out all the pre-processing steps to the image. The engine would return text that was wrong or had a lot of extra characters that were not in the picture.

#### ***2.4.4 Combining text and pie slices***

Linking a label from the text reader to a calculated ratio from the pie is the third and last step before returning the results to the user. To do so the program needs to go through each label and check on its left side if it has a small icon of a colour that matches one of the colours in the pie. To find the colour icon the positional data from the label is used, take the positional data, and start from the left upper corner and move down quarter of the height of the word and then search left for pixels in a darker colour. If enough coloured pixels are found, then that colour is compared to one of the pie colours and if there is a match the label is assigned to the slice. Once every label has been assigned one of the slices found in the pie the data is passed to a function that calculates the value of each slice from the sum passed in with the request relative to the ratio found in the image. Then the data is packaged into a Pandas DataFrame to be returned to the API.

### **2.5 API: application programming interface**

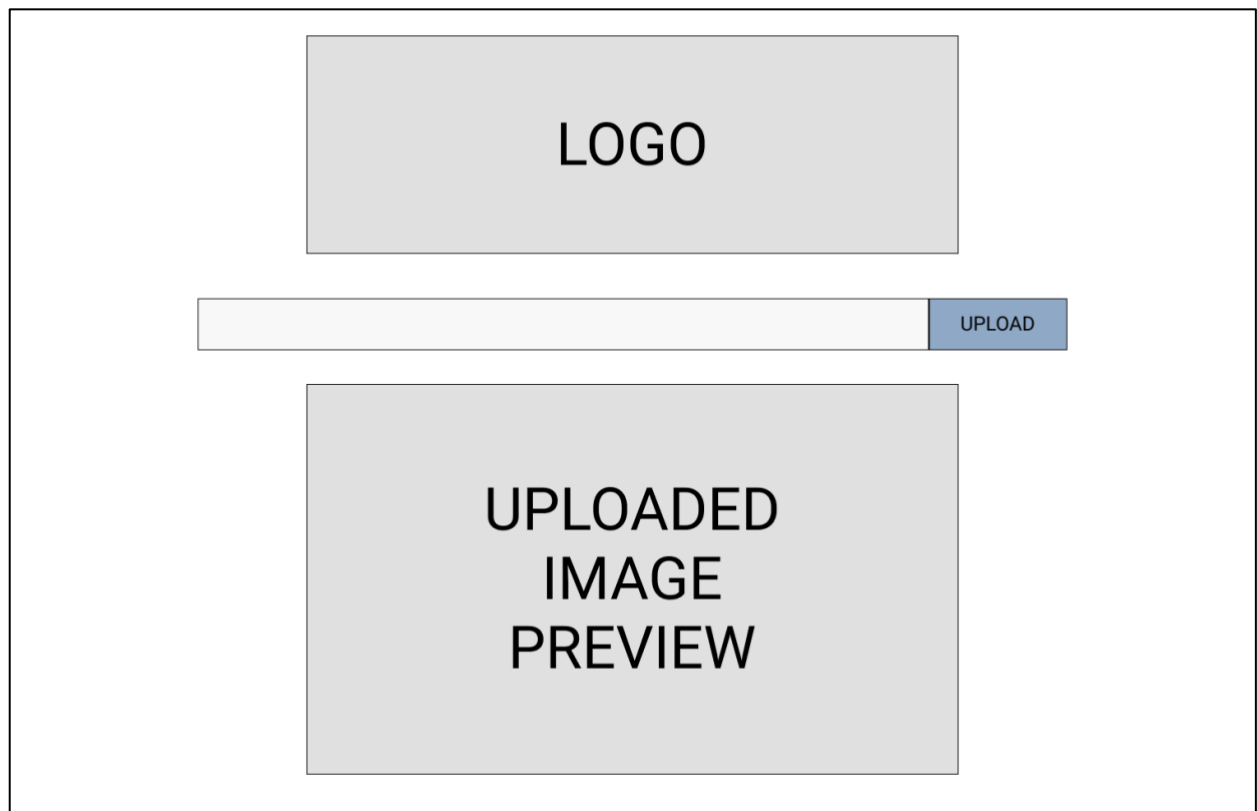
Having an API makes this project more versatile and can be used with many more systems than just the frontend provided in this project. Some company or personal user could use this system with their system without hosting it on their own servers. The reason behind this was to have the system more versatile and not be limited by the frontend the team creates. The API parameters are described in a figure below.

Figure 8: API

API
<p><b>files:</b> a parameter that takes an image of the format 'png', 'jpg' or 'jpeg'</p> <p><b>selectedFormat:</b> a parameter of type string that signifies the requested file type to be returned from the program. Available formats are: 'csv' 'xlsx' and 'json'. Default is 'json'</p> <p><b>selectedChart:</b> a parameter of type string that signifies whether the graph is 'pie' or 'stacked'</p> <p><b>selectedSum:</b> a optional parameter of type int. If supplied then additional column will be added which is the relative value of each slice compared to the total sum given. I.e. if a slice is 25% of a graph and selectedSum is 300 then that slice will be valued at 75</p>

## 2.6 Wireframes of the frontend

Figure 9: Frontend wireframe 1



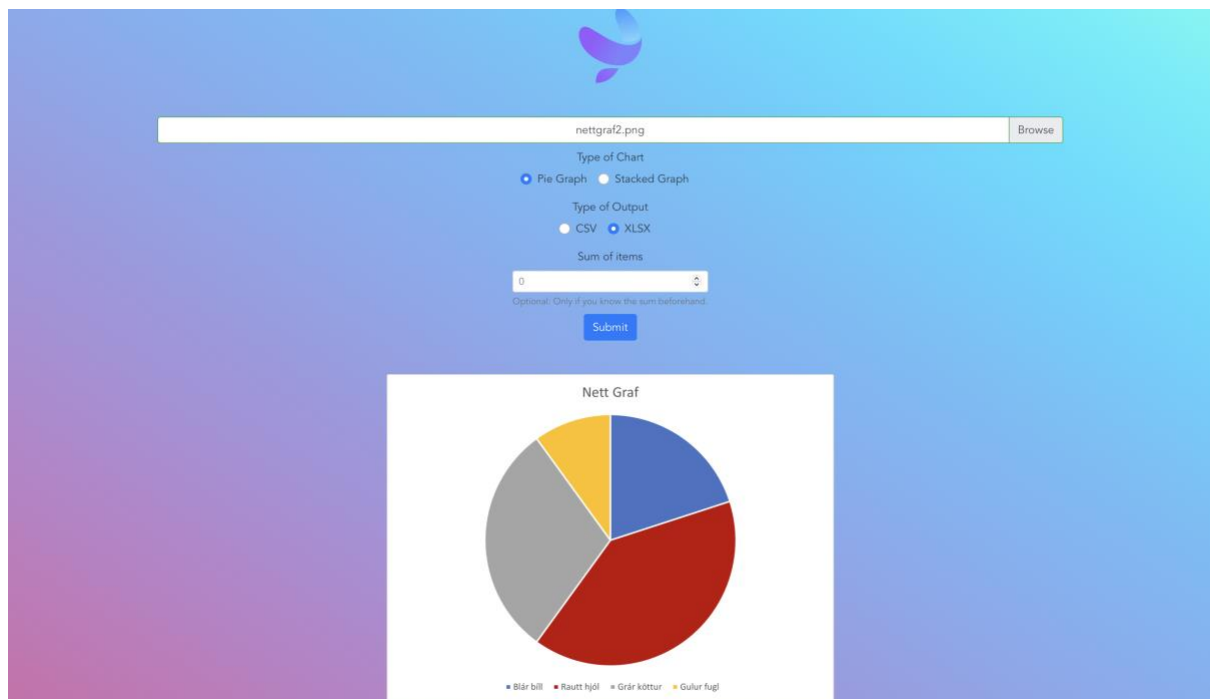


Due to most of the project is to create a web service instead of a full application, the frontend will be very minimal, only as an example entry point to the backend API which handles extracting the data from the graphs.

In developing the project, the team discovered that it would be simpler and would enable a more feature rich program by adding some settings. Choosing if the user is uploading a pie graph or a stacked bar graph was a feature to be able to process stacked bar graphs without any machine learning algorithm detecting whether a pie graph or stacked graph was uploaded. Choosing a file format was also added so the user had the option of choosing what file format suited their needs best. Adding the sum field was a feature added if the user knew what the total sum of the graph was then it would be included in the calculations and file returned to the user.

Final look of the frontend can be seen in the figure below.

*Figure 10: Final look of the frontend*



The screenshot displays a web application interface with a blue gradient background. At the top center is a purple logo. Below it is a file input field containing the text 'nettgraf2.png' and a 'Browse' button. Under the file field are two radio buttons for 'Type of Chart': 'Pie Graph' (selected) and 'Stacked Graph'. Below these are two radio buttons for 'Type of Output': 'CSV' and 'XLSX' (selected). A 'Sum of items' input field is set to '0', with a small note below it: 'Optional. Only if you know the sum beforehand.' A blue 'Submit' button is positioned below the sum field. In the center of the page is a white box titled 'Nett Graf' containing a pie chart with four segments: blue, red, grey, and yellow. A legend at the bottom of the chart identifies the segments: 'Blár billi' (blue), 'Rautt hjól' (red), 'Grár köttur' (grey), and 'Gulur fugl' (yellow).

## 2.7 Testing

### 2.7.1 Unit testing

For unit testing a framework called pytest is used. Pytest makes testing easy and simple, you make python documents that end with “\_test.py” and when you run pytest on the project it looks for all files with that name in every subdirectory and runs the test and gives a report on how many tests succeeded and how many failed. This is very handy since if all tests are

written and someone makes a change one can simply run the command “pytest” in the terminal and check if everything still works as expected, given the tests have good coverage. This framework is also great since it is quite easy to use with GitHub actions. It runs the tests automatically when trying to merge to the main branch. With pytest action setup on GitHub it makes sure that every time a pull request is created to merge to the main branch the code always works. The team set a rule so a merge cannot happen to the main branch if the tests do not all succeed, and it is not possible to push to the main branch remote repository. Having these types of tests and safety measures helps us minimize the risk of deploying code that does not work.

Test coverage of the unit tests is variable. Every function has at least one test to make sure it runs correctly. Some have more to try and cover edge cases and make the tests more robust. With the code in this project there are few inputs and the image processing either works or it does not. So, it can be rather difficult to write tests for edge cases since a lot of the functions and libraries that are being used are like black boxes and it is nearly impossible to create edge case tests without creating them by trial and error.

*Table 1: Unit test coverage*

<b>Name</b>	<b>Statements</b>	<b>Miss</b>	<b>Coverage</b>
backend/processgraph/__init__.py	1	0	100%
backend/processgraph/graph2data.py	54	7	87%
backend/processgraph/image_proc.py	87	11	87%
backend/processgraph/percentage_finder.py	41	0	100%
backend/processgraph/textread.py	81	9	89%
<b>Total</b>	<b>264</b>	<b>27</b>	<b>90%</b>

In Table 1: Unit test coverage with data exported from pytest with the coverage of the image processing module we wrote.

### **2.7.2 Accuracy testing**

For accuracy testing a python script was written to test a set of chart images where the data to create those charts are known and used to measure the accuracy of the program. The set that was created contains both charts that have good charts with large slices in a pie and each slice

has a distinctly different colour from the rest, so the program does not confuse two slices together. But the set also contains data with very many slices and colours that are alike to test and measure how the program performs with data that is known to be challenging. Doing these kinds of tests helps determine if changes that are made to the program have negative or positive or even negligible impact on the performance of the program. Analysing performance is crucial when developing this kind of program since the results can be hard to predict. It can be evaluated with the formula

*Equation 1: Accuracy metric*

$$\Phi = 1 - \sum_{i=1}^{N_s} |\Delta X_i|$$

where  $N_s$  number of all slices in a pie and  $\Delta X$  is the normalized difference between the actual size and the calculated size of a given slice. The team created this formula and chose to use this as a metric to determine accuracy. The team thought of more complex ways to calculate the accuracy, but the conclusion was that this metric was simple and worked as intended. To evaluate if a calculation by the program is good or not, this formula does a decent job. If the evaluation is 1, then the result was perfect and if the result is lower it means that the calculation was not perfect and the further the result goes below 1 the worse the calculations were to the true value.

Number	Labels	Accuracy
1	12_entries	0,7
2	fiveshadesofRED_20values_appart	0,8
3	bogatest	1
4	fiveshadesofRED_10values_appart	-0,17
5	fiveshadesofgray	0,6
6	rokk_stig	0,98
7	8_entries	0,96
8	11_entries	0,73
9	10_entries	0,88

In the table above we have 9 different types of pie charts created in excel and the accuracy of our program which is calculated with Equation 1: Accuracy metric mentioned above.

Conclusions that can be drawn from the tests are that colours that are very similar in value do

not lead to good results and when the number of slices become close to 10 you start losing accuracy, given that each slice is similar in size. One entry can be seen becoming negative and the explanation behind that the error between each slice can be so big that the total sum of errors is greater than one and that leads to a result that becomes negative. This is one of the downsides of the formula, but the results still tell us that this graph performed extremely badly in our program. Even though accuracy measurements are not supposed to become negative the usability of this accuracy metric is not obsolete. All the images tested can be found in Test images and the data use can be found in Test data in the appendix.

### 3 Risk analysis

There are many risk factors to be taken into consideration when it comes to a project of this scale. Many things can go wrong so to minimize those risk the team did a detailed risk analysis. When new risks are discovered, they are assigned two variables, probability, and severity both on the scale 1-5. Then those two variables are multiplied together to produce the total risk points of the risk. These total risk points are then used to evaluate the expected impact to the project for each risk separately. For each risk there must be a way to minimize its likelihood of happening, and how to respond to mitigate the effects on the project if the risk becomes a reality.

$$\text{likelihood} * \text{severity} = \text{total risk points}$$

Table 2: Risk analysis

Risk	Description	Mitigation	Prevention	P	S	T R
<b>Complexity of project is too high</b>	The complexity of the project exceeds the skill level of the team.	Lower the number of necessary criteria for definition of done.	Continuous analysis of the complexity of the project and keeping the project on track.	3	5	15
<b>Person responsible:</b> Hákon						
<b>Event log:</b>						
28/02/22: Teacher at RU explained the complexity of the original project and convinced the team that it was too much for brief time and lack of experience of the team.						
<b>Workload in other courses</b>	Workload in other courses causes the team to have less time for the project.	Try to keep up with the other courses.	Make up for lost time once workload decreases.	4	3	12
<b>Person responsible:</b> Finnbogi						
<b>Event log:</b>						

28/03/22: Final exams put a total halt on the project.						
<b>Sickness</b>	A team member gets sick and cannot work on the project.	Make up for lost time by putting in hours outside of the set schedule.	Constant communication between team members so everyone knows who is doing what.	4	2	8
<b>Person responsible:</b> Gísli						
<b>Event log:</b>						
21/02/22: Gísli got COVID and had to work from home. 02/05/21: Gísli was sick and not able to participate						
<b>Unexpected bugs</b>	Bugs that occur that will take time to resolve.	Test everything before pushing to main branch.	Pair up to figure out the bug and resolve it.	4	2	8
<b>Person responsible:</b> Hákon						
<b>Event log:</b>						
21/03/22: Bug while finding percentages, the colours in the labels had impact on the result. 06/04/22: Colour indicator of labels was detected as text. 14/04/22: Random spots on image were being treated as text. 04/05/22: Docker version of Pytesseract had issues with reading some text.						
<b>Stuck on a problem</b>	A team member gets stuck on a problem and cannot continue until it is resolved.	The team discusses the problem and solves it together.	Regular discussions of the project and what each member is doing, keeping everyone on the team informed on each other's tasks.	4	2	8
<b>Person responsible:</b> Finnbogi						
<b>Event log:</b>						
15/03/22: the size and complexity of the OpenCV library halted progress. 14/04/22: Pytesseract had some unexplainable issues which caused a headache.						
<b>Loss of code</b>	A part of the code gets lost due to any reason	The team writes the code back from memory and tries to bring back the functionality	Constant commits to a version control system.	2	4	8
<b>Person responsible:</b> Gísli						
<b>Event log:</b>						
No events.						
<b>A dependency gets deprecated</b>	A library we are using gets updated and breaks our program.	Read the docs for the updated library and fix our code to work with the latest version.	Have a fixed version number for each dependency.	1	5	5
<b>Person responsible:</b> Hákon						

<b>Event log:</b>						
No events.						
<b>Absence of team member</b>	Team member cannot attend meetings due to unforeseen circumstances.	Absent team member tries to remotely partake in the meeting via discord or teams.	Keep in mind dates and times of meetings before making other plans.	3	1	3
<b>Person responsible:</b> Finnbogi						
<b>Event log:</b>						
No events.						
<b>A dependency has vulnerability</b>	A vulnerability is discovered in a library being used in the program	Update to the latest version of the library if that vulnerability has been patched.	The team keeps up to date on vulnerabilities of the libraries in use.	1	2	2
<b>Person responsible:</b> Gísli						
<b>Event log:</b>						
No events.						

## 4 Product backlog

Table 3: Product backlog

Product backlog item	Priority	Estimated hours	Status
<b>Set up and planning</b>			
Create a general plan for the project.	A	20	Done
Have a meeting with the CEO of Grid.	A	5	Done
Set up our equipment at the Grid offices.	A	5	Done
Research what libraries to use for the project.	A	5	Done
Get acquainted with the team at Grid.	B	5	Done
<b>Frontend</b>			
Decide on frontend framework.	A	15	Done
Limit uploads to only accept images.	A	10	Done
Add a functionality to upload a photo.	A	15	Done

Add a preview of the currently selected photo.	B	10	Done
Add ability to pick chart type.	B	10	Done
Add ability to pick output format.	B	10	Done
Setup docker image for frontend.	B	10	Done
Host frontend on AWS EB.	B	20	Done
<b>Backend API</b>			
Decide on backend framework.	A	15	Done
Set up a basic API.	A	20	Done
Create API endpoint to receive images.	A	15	Done
Backend returns xlsx file.	C	5	Done
Backend returns csv file.	C	5	Done
Backend returns JSON string.	A	5	Done
Validate file type received.	A	5	Done
Setup docker image for backend.	B	20	Done
Host backend on AWS EB.	B	20	Done
<b>Image processing</b>			
Functionality to detect circles.	A	30	Done
Functionality to detect rectangles.	B	30	Done
Mask out certain shapes.	A	20	Done
Upscale an image under a certain size.	B	10	Done
Remove noise from an image.	A	5	Done
Threshold an image for easier text reading.	A	5	Done
<b>Text detection and reading</b>			
Detect location and contents of text in image.	A	40	Done
Locate nearby colours from text location.	A	50	Done

Assign contents of text to a colour value.	A	25	Done
Combine adjacent words.	B	20	Done
Read axis of a regular bar chart	C	100	Unfinished
<b>Colour detection / Percentage finder</b>			
Find all non-white pixels in an image.	A	40	Done
Group together pixels of the same colour.	A	40	Done
Calculate the ratio of each group of coloured pixels.	A	20	Done
Connect labels and percentages based on shared colour.	A	30	Done
Implement detection for regular bar graphs	C	50	Unfinished
<b>Progress report and presentation</b>			
Write a risk analysis.	A	50	Done
Write a progress report.	A	60	Done
Write a system Overview.	A	60	Done
Create a user guide.	A	5	Done
Crat an operating manual.	A	20	Done
Create presentations.	A	60	Done
<b>Documenting and improving code</b>			
Documenting the code.	A	20	Done
Write unit tests for image processing.	B	10	Done
Write unit tests for text detection.	B	10	Done
Write unit tests for colour detection.	B	10	Done
Improving code.	A	100	Done



## 5 Progress report

In this section there is an overview of all the sprints the group performed and the planning, backlog, review, burndown, and retrospective for each sprint. In the burndowns the y-axis is displayed in weeks and days, 1 week is 5 days and 1 day is 8 hours.

### 5.1 Project burndown

Figure 11: Project burndown

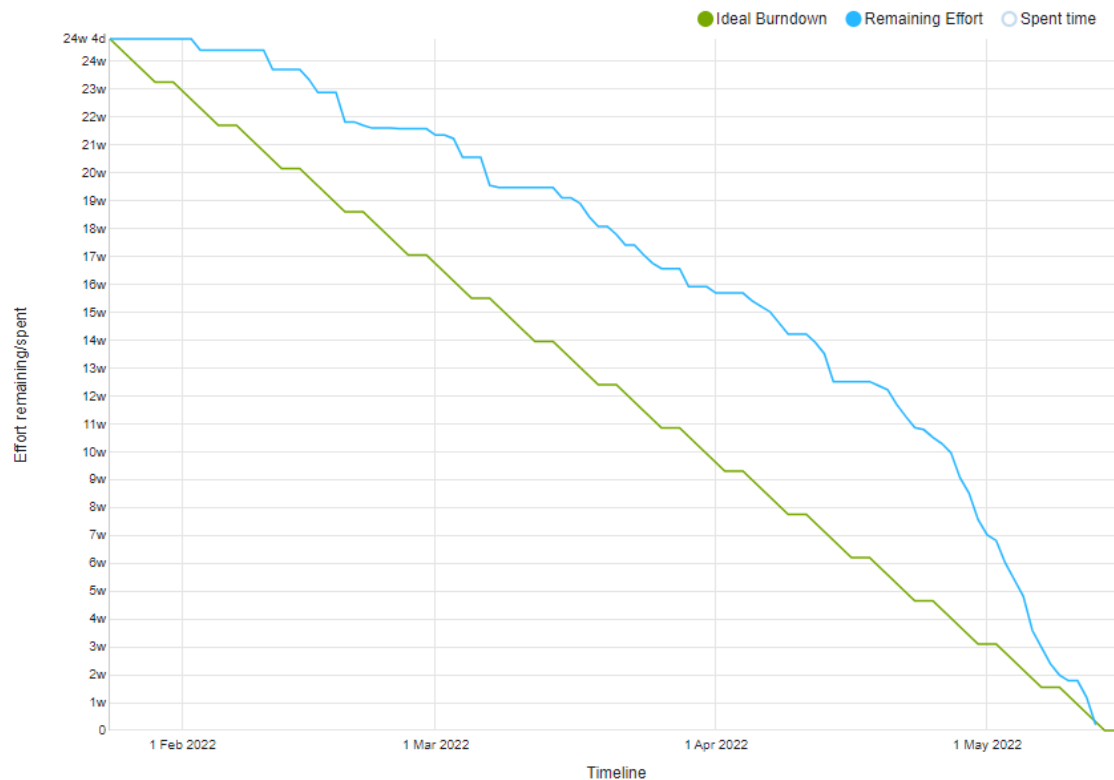
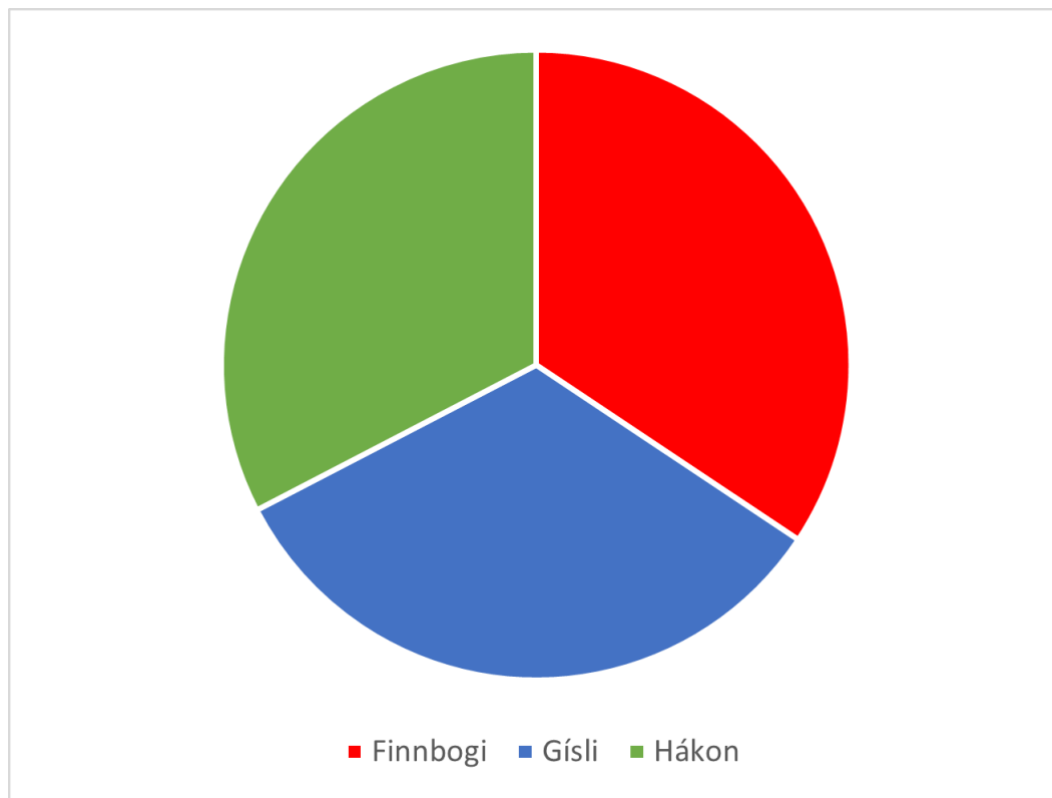


Table 4: Total time spent

Team member	Time spent
Finnbogi Jakobsson	336 hours
Gísli Þór Gunnarsson	323 hours
Hákon Hákonarsson	319 hours
<b>Total</b>	<b>979 hours</b>

Figure 12: Work distribution



More detailed charts on work distribution can be found under in Work distribution by work type under the appendix.

## 5.2 Sprint overview

### 5.2.1 *Sprint 0*

The first sprint was focused on creating a plan for the project and making everything ready to begin working on the project, so we decided to call it sprint 0.

### 5.2.2 *Sprint 1*

This sprint focused researching and deciding frameworks and tools as well as doing the report for the first status meeting.

### 5.2.3 *Sprint 2*

This sprint focused on getting started coding by setting up a basic front- and backend for the product as well as establishing a connection between the two.

### 5.2.4 *Sprint 3*

During this sprint the team focused on getting started coding by setting up a basic front- and backend for the product as well as establishing a connection between the two.

### **5.2.5 *Sprint 4***

This sprint was short because Reykjavík University's final exams were during the second week of this sprint. Nevertheless, satisfactory progress was made in the project during the first week and a functional version was developed which could detect all the colour ratios in a pie chart.

### **5.2.6 *Sprint 5***

During this sprint the team managed to finish implementing the pie chart algorithm although some more testing is required. This was done by detecting the labels and assigning them to the correct pie slices.

### **5.2.7 *Sprint 6***

During this sprint, the team added more image processing to make it easier for the program to read text and calculate the percentages of each slice by removing unneeded elements from the images.

### **5.2.8 *Sprint 7***

In this final sprint the team added a function to detect stacked bar charts, finished creating unit tests and then implemented a code freeze on the 5<sup>th</sup> of May. During the last week and a half, the team focused on finishing the report and presentation of the project.

## **6 Known bugs**

### **6.1 Text algorithm**

Most serious bug is the text detection algorithm. It sometimes adds extra characters to labels and in other cases does not detect the label, so we had to place a “default” label which is just a number.

### **6.2 Dark coloured background**

The algorithm assumes that the background is white. Which means that all images with any background colour other than white will lead to very incorrect results.

### **6.3 Size check of uploaded file**

There is not a check for file size in the backend so a user could generate a very large image and what would happen is unknown since it has never been tested.

## **6.4 Graphs that are not made in Excel**

All the graphs used in the project are created in Excel, so the results are biased towards excel. So other graphs, especially those who do not have a similar format to Excel, will not perform as well with the system.

## **6.5 GET [transmo-api.grid.is/finn/bogi](https://transmo-api.grid.is/finn/bogi)**

We do not know why but it returns {"Finn": "bogi"}.

## **6.6 Removal of colour icons**

The removal of colour icons next to labels in images does not always work. It is hard to finetune.

# **7 Conclusion**

## **7.1 Final product**

The team has created a working proof of concept that they are proud of. The product is hosted with the frontend and backend separately and everything works end to end, which is a great accomplishment, especially since the image processing works on graphs and returns usable data. The final product can detect both stacked bar chart and pie charts with good accuracy when good images are used as input. A user can request through the frontend to receive a csv (comma separated values) or a excel spreadsheet and the user can supply the program a total sum of the whole graph and the program will calculate the value of each slice. The final design for the frontend is simple but functional as can be seen in Figure 10: Final look of the frontend in chapter 2.6. For enterprise users or companies, the program also has an API to receive requests to the image processing directly which could be a sold in the future if other companies have interest in the product. The parameters on the API can be seen in Figure 8: API in chapter 2.5.

## **7.2 Retrospective**

This chapter serves as a retrospective for the whole project. After finishing the project, the team sat down and had a conversation about what went well and what could have gone better. The team is overall satisfied with the project although there were some difficulties during the development. There are some things the team would like to have done differently but due to time constraints and lack of experience with computer vision and machine learning the team had to find a more reasonable approach.

### ***7.2.1 What went well***

The team was satisfied with how quickly the end-to-end flow was setup which then was continuously improved upon. Setting up continuous deployment went smoothly thanks to assistance for employees at Grid which helped the team set up AWS Elastic Beanstalk deployment of the front- and backend services which used the latest build from the teams' repositories master branches. The communication in the team was good, by having regular meetings and always staying in touch outside of those meetings, the team managed to keep a good flow of the project going.

### ***7.2.2 What could have gone better***

In the beginning of the project, it took some time for the team to get started on the project, this loss of valuable time could have been avoided if the team had looked for guidance from more experienced personnel earlier on in the lifespan of the project. The team would have liked to use the product backlog more properly and kept a better track on their time spent, there were occasions where the team implemented a feature and forgot to add the time spent on the task which had to be fixed later.

### ***7.2.3 Future development***

As the finished product from this project is a prototype there is still a lot that can be done, implementing the ability to handle more types of graphs is at the top of the list. Adding a neural network to the solution that would be able to detect the type of graph given would improve the user experience and reduce the number of inputs from the user. There can some improvements be made in the text reading portion of the project since there are still some issues that the team didn't manage to solve, if that is more pre-processing of the image or adjusting some parameters in the Pytesseract function calls which could provide better results.

## 8 Appendix

### 8.1 Sprints

#### 8.1.1 Sprint 0

*From the 27th of January 2022 to 6th of February 2022*

The first sprint was focused on creating a plan for the project and making everything ready to begin working on the project, so we decided to call it sprint 0.

##### 8.1.1.1 Sprint planning

- What? Get settled in the Grid offices and meet with the product owner to discuss the requirements of the software.
- How? Going to the Grid offices and setting up our work area and meeting the product owner.
- Who? All team members.

##### 8.1.1.2 Sprint backlog

*Table 5: Sprint 0 backlog*

<b>Sprint backlog item</b>	<b>Prioritization</b>	<b>Estimated hours</b>	<b>Finished</b>
Create a general plan for the project.	A	20	X
Have a meeting with the CEO of Grid.	A	5	X
Set up our equipment at the Grid offices.	A	5	X
Get acquainted with the team at Grid.	B	5	X

##### 8.1.1.3 Sprint Review

All went well and the team is set up in the offices, the team has a good idea on what to do for the project after discussing requirements with the product owner

##### 8.1.1.4 Sprint timesheet

*Table 6: Sprint 0 time spent*

<b>Team member</b>	<b>Time spent</b>
Finnbogi Jakobsson	12 hours

Gísli Þór Gunnarsson	15 hours
Hákon Hákonarsson	13 hours
<b>Total</b>	40 hours

#### 8.1.1.5 Sprint retrospective

Everything worked very well, the one thing that the team could improve is taking advantage of the in-house resources at the grid office and of the knowledge the people who work there have.

### 8.1.2 Sprint 1

*From the 7th of February 2022 to 20th of February 2022*

This sprint focused researching and deciding frameworks and tools as well as doing the report for the first status meeting.

#### 8.1.2.1 Sprint backlog

*Table 7: Sprint 1 backlog*

Sprint backlog item	Prioritization	Estimated hours	Finished
Decide on backend framework.	A	15	X
Decide on frontend framework.	A	15	X
Research what libraries to use for the project.	A	5	X
Write a risk analysis.	A	20	X
Write a progress report.	A	20	X
Write a system Overview.	A	20	X
Create presentations.	A	20	X

#### 8.1.2.2 Sprint planning

- What? To get started on the report and decide on frameworks and set up the development environments needed.

- How? Discuss and research what frameworks would work best for this type of project and decide on how to store the product backlog and user stories.
- Who? All team members.



### 8.1.2.3 Sprint review

The team decided to use a Vue.js framework to manage the frontend which will be connected to a Django backend. OpenCV was chosen due to it being open source and having good documentation and community support. GitHub was chosen for the version control and to manage the product backlog. The team started work on the report with focus on the risk analysis, project description and the product backlog.

### 8.1.2.4 Sprint burndown

Figure 13: Sprint 1 burndown

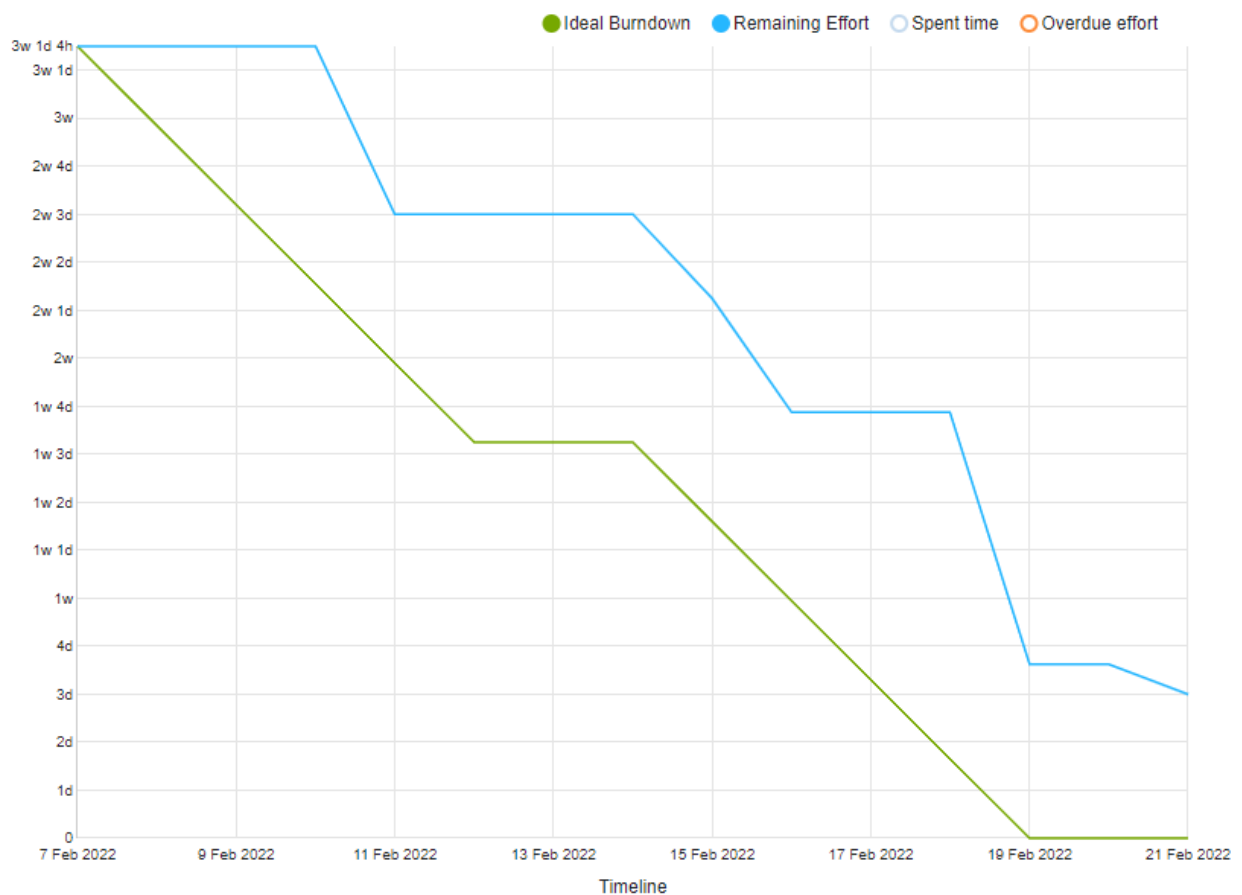


Table 8: Sprint 1 time spent

Team member	Time spent
Finnbogi Jakobsson	49 hours
Gísli Þór Gunnarsson	37 hours
Hákon Hákonarsson	46 hours

<b>Total</b>	132 hours
--------------	-----------

#### 8.1.2.5 *Sprint retrospective*

The sprint went fine overall. After the first progress meeting, the team tried to fix all the mistakes which were present in the report and presentation and decided to continue the report in Word instead of Overleaf.

### 8.1.3 *Sprint 2*

*From the 21st of February 2022 to 6th of March 2022*

This sprint focused on getting started coding by setting up a basic front- and backend for the product as well as establishing a connection between the two.

#### 8.1.3.1 *Sprint planning*

- What? To set up a basic front- and backend and establish a connection between them with HTTP requests.
- How? By reading the required documentation to familiarise ourselves with the frameworks and then implementing a basic front- and backend.
- Who? All Team members.

#### 8.1.3.2 *Sprint backlog*

*Table 9: Sprint 2 backlog*

<b>Sprint backlog item</b>	<b>Prioritization</b>	<b>Estimated hours</b>	<b>Finished</b>
Add a functionality to upload a photo.	A	15	X
Limit uploads to only accept images.	A	10	X
Add a preview of the currently selected photo.	B	10	X
Set up a basic API.	A	20	X
Create API endpoint to receive images.	A	15	X
Validate file type received.	A	5	X
Backend returns csv file.	C	5	X

### 8.1.3.3 *Sprint review*

The team managed to complete all tasks except “Setup the machine learning OpenCV environment” due to rescopeing of the project. In the second half of the sprint Hákon spoke with Gylfi, a professor at RU to get some pointers on how to approach the computer vision part of the project and got some interesting and quite depressing feedback. Gylfi pointed out that doing computer vision “from scratch” is way too ambitious for a BSc project. He made us realise that this will be harder than we anticipated. He recommended that we would scale down and simplify the project and gave us some ideas on how to do so.

- Start with analysing pie-charts and use colour histograms and use an algorithm that counts the number of pixels in each slice. That way we can compare the size of each slice and give a percentage back in a table.
- If we were to continue with OpenCV or some computer vision. Then at least implement some UI that makes the user draw bounding-boxes around the Y-axis, X-axis, and the chart itself. That would at least simplify one big step of the process.
- Things that Gylfi mentioned to look at: RCNN, Faster RCNN, OCR (Tesseract OCR), YOLO algorithm, segmentation of images by colour, colour histogram and some algorithm called fill-out I think that checks if neighbouring pixel is the same colour.

### 8.1.3.4 Sprint burndown

Figure 14: Sprint 2 burndown

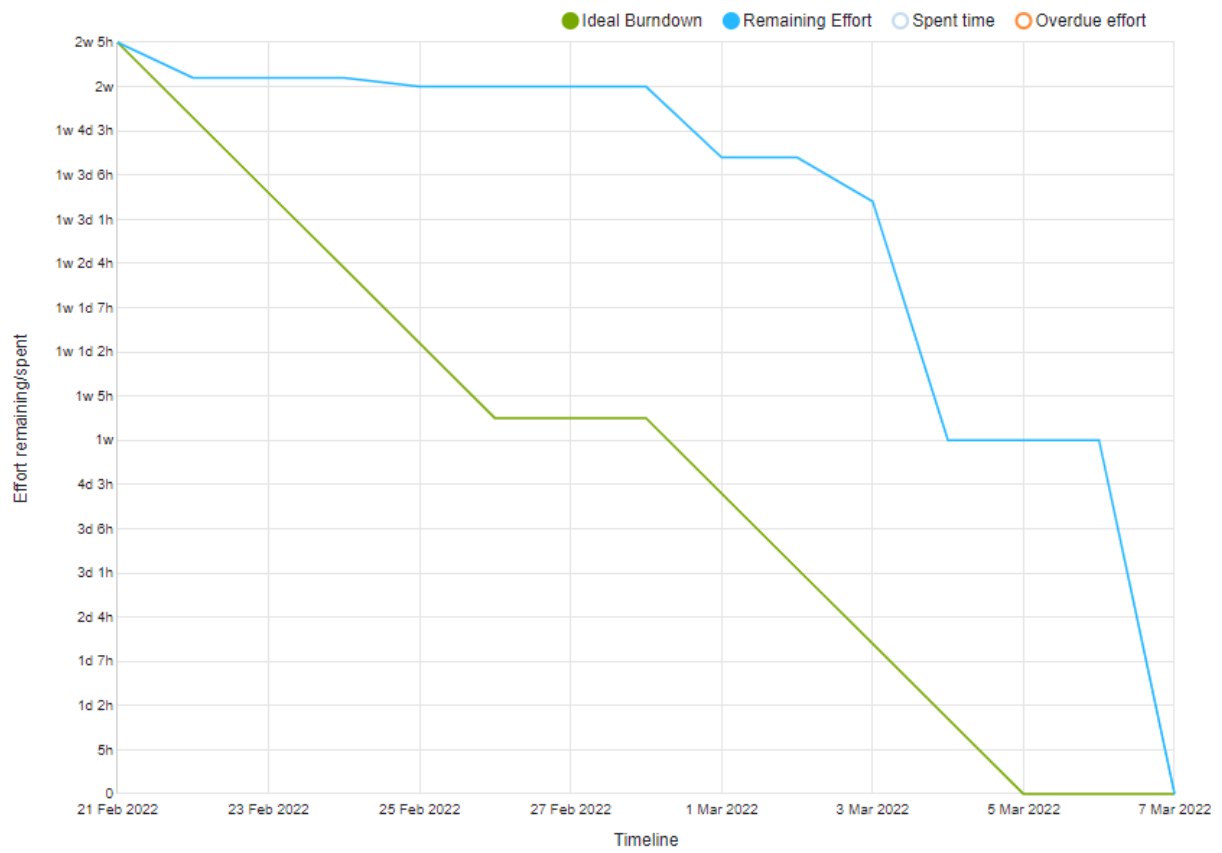


Table 10: Sprint 2 time spent

Team member	Time spent
Finnbogi Jakobsson	28 hours
Gísli Þór Gunnarsson	12 hours
Hákon Hákonarsson	40 hours
<b>Total</b>	80 hours

#### 8.1.3.5 Sprint retrospective

The sprint did not go as planned. After the realization that the project was too complex the morale of the team took a big hit which caused the efficiency of the team to dramatically decrease. The team will try to improve this as soon as possible too waste as little time as possible.

#### 8.1.4 Sprint 3

*From the 7th of March 2022 to 20th of March 2022*

For sprint 3 the team wanted to implement CI/CD and get the backend hosted on AWS Elastic Beanstalk. To do so we had help from an employee at Grid who has expertise in these matters.

##### 8.1.4.1 Sprint planning

- What? To set up CI/CD for the backend and be able to extract data from pie graphs.
- How? Get assistance from an expert from Grid for setting up CI/CD and read up on algorithms that can count numbers of colours of each pixel.
- Who? All Team members.

##### 8.1.4.2 Sprint backlog

Table 11: Sprint 3 backlog

Sprint backlog item	Prioritization	Estimated hours	Finished
Find all non-white pixels in an image.	A	40	
Setup docker image for backend.	B	20	X

Host backend on AWS EB.	B	20	X
Write a risk analysis.	A	10	X
Write a progress report.	A	10	X
Create presentation.	A	10	X

#### 8.1.4.3 Sprint review

The team managed to implement CI/CD and the backend is now hosted on AWS Elastic Beanstalk which always has the latest version of the service and is open to HTTP requests. Making a simple pie-chart detection algorithm must be moved over to sprint 4 since the team was not able to finish it, but some progress was made.

#### 8.1.4.4 Sprint burndown

Figure 15: Sprint 3 burndown

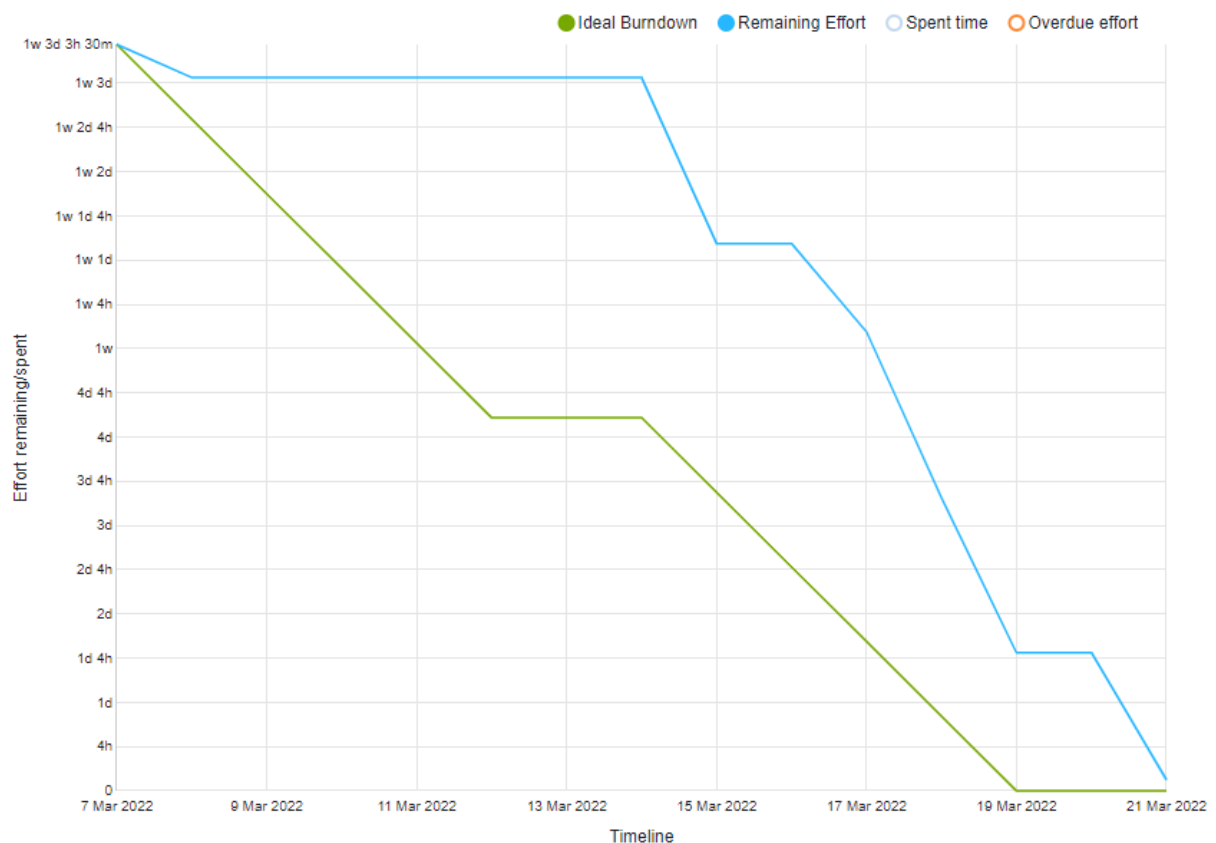


Table 12: Sprint 3 time spent

Team member	Time spent
Finnbogi Jakobsson	31 hours
Gísli Þór Gunnarsson	28 hours
Hákon Hákonarsson	5 hours
<b>Total</b>	64 hours

#### 8.1.4.5 Sprint retrospective

At the start of the sprint the team was still suffering from morale loss after last sprint but that was fixed during the sprint, and everything should be getting back on track.

#### 8.1.5 Sprint 4

*From the 21st of March 2022 to 3rd of April 2022*

Sprint 4 was short because Reykjavík University's final exams were during the second week of this sprint. Nevertheless, satisfactory progress was made in the project during the first week and a functional version was developed which could detect all the colour ratios in a pie chart.

##### 8.1.5.1 Sprint planning

- What? Develop a program which takes an image of a pie chart as an input and output the fraction of each slice.
- How? By reading the required documentation to familiarise ourselves OpenCV and then developing a program to read pie charts.
- Who? All Team members.

### 8.1.5.2 Sprint backlog

Table 13: Sprint 4 backlog

Sprint backlog item	Prioritization	Estimated hours	Finished
Find all non-white pixels in an image.	A	40	X
Group together pixels of the same colour.	A	40	X
Calculate the ratio of each group of coloured pixels.	A	20	X

### 8.1.5.3 Sprint review

All tasks on the product backlog were completed but some improvements can still be made.

### 8.1.5.4 Sprint burndown

Figure 16: Sprint 4 burndown

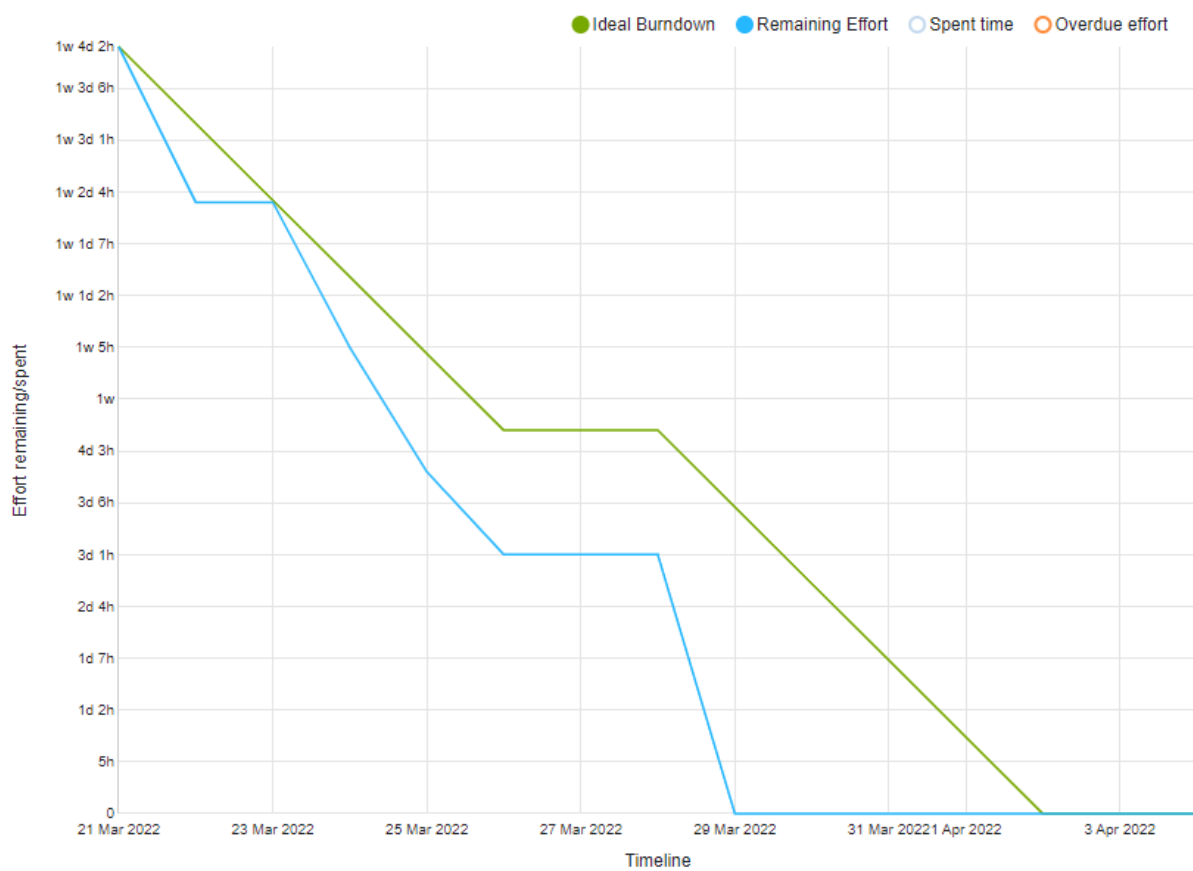




Table 14: Sprint 4 time spent

Team member	Time spent
Finnbogi Jakobsson	14 hours
Gísli Þór Gunnarsson	26 hours
Hákon Hákonarsson	31 hours
<b>Total</b>	71 hours

#### 8.1.5.5 Sprint retrospective

The overall sprint went well given the reduced time available from the team due to final exams.

#### 8.1.6 Sprint 5

*From the 4th of April 2022 to 17th of April 2022*

During this sprint the team managed to finish implementing the pie chart algorithm although some more testing is required. This was done by detecting the labels and assigning them to the correct pie slices.

##### 8.1.6.1 Sprint planning

- What? Finish the developing the pie chart algorithm.
- How? By creating a way to detect labels and combining them with the ratios of the slices
- Who? All Team members.

### 8.1.6.2 *Sprint backlog*

Table 15: *Sprint 5 backlog*

<b>Sprint backlog item</b>	<b>Prioritization</b>	<b>Estimated hours</b>	<b>Finished</b>
Detect location and contents of text in an image.	A	40	X
Locate nearby colours from text location.	A	50	X
Assign contents of text to a colour value.	A	25	X
Connect labels and percentages based on shared colour.	A	30	X

### 8.1.6.3 *Sprint review*

The team completed all tasks although some could be improved on in future sprints.

#### 8.1.6.4 Sprint burndown

Figure 17: Sprint 5 burndown

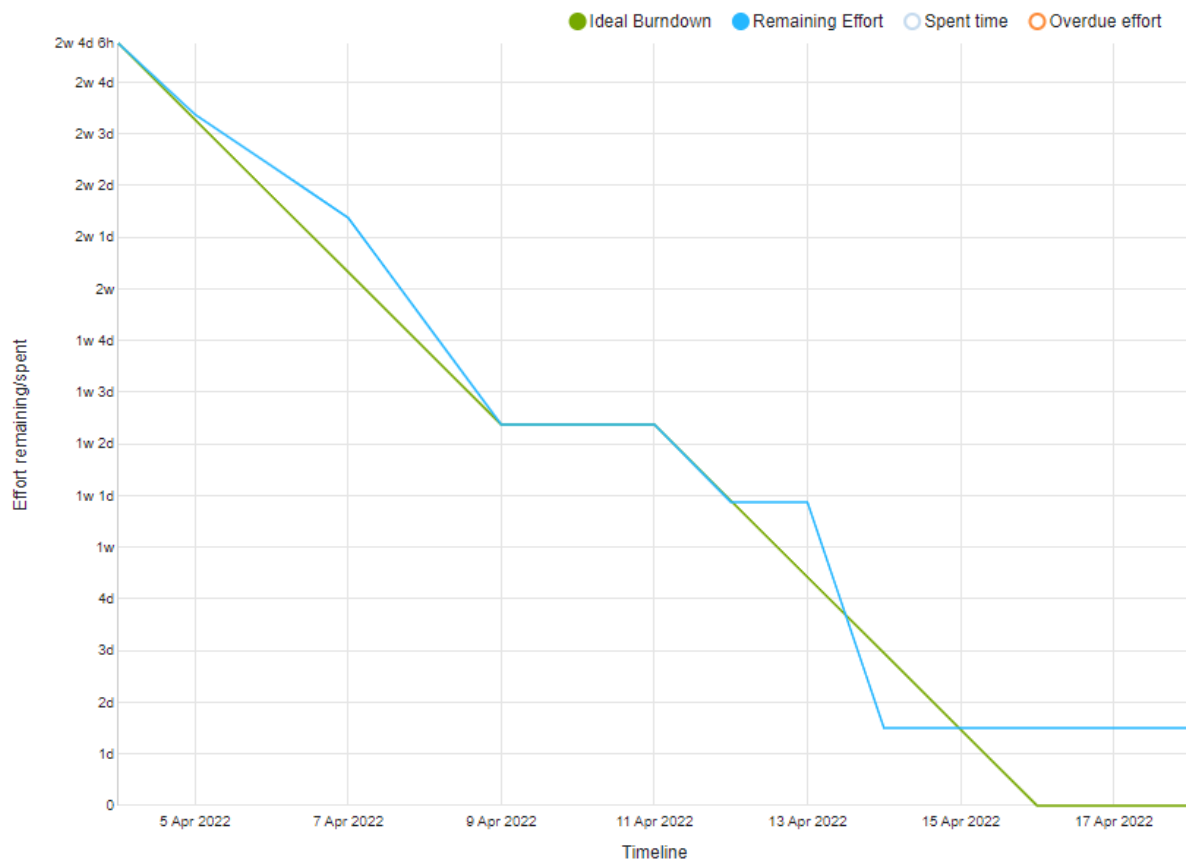


Table 16: Sprint 5 time spent

Team member	Time spent
Finnbogi Jakobsson	10 hours
Gísli Þór Gunnarsson	54 hours
Hákon Hákonarsson	52 hours
<b>Total</b>	<b>116 hours</b>

#### 8.1.6.5 Sprint retrospective

Had some issues with the Pytesseract library not working as expected but managed to figure out a solution which will be improved upon in future sprints.

### 8.1.7 Sprint 6

*From the 18th of April 2022 to 1st of May 2022*

During this sprint, the team added more image processing to make it easier for the program to read text and calculate the percentages of each slice. This was done two separate ways, by first removing everything from the image sent to the colour detection function except the pie or stacked bar and then calculating the percentages of each colour. The second function implemented was to remove the pie from the image and then read the text to reduce the risk of errors.

#### 8.1.7.1 Sprint planning

- What? Implement and improve image processing to make colour and text detection easier.
- How? By reading relevant documentation and trying different approaches to processing.
- Who? All Team members.

#### 8.1.7.2 Sprint backlog

*Table 17: Sprint 6 backlog*

<b>Sprint backlog item</b>	<b>Prioritization</b>	<b>Estimated hours</b>	<b>Finished</b>
Functionality to detect circles.	A	30	X
Functionality to detect rectangles.	B	30	X
Mask out certain shapes.	A	20	X
Remove noise from an image.	A	5	X
Threshold an image for easier text reading.	A	5	X
Improving code.	A	80	
Upscale an image under a certain size.	B	10	X

#### 8.1.7.3 Sprint review

During this sprint all the tasks except finishing improving the code since there were still some known bugs which needed fixing.

#### 8.1.7.4 Sprint burndown

Figure 18: Sprint 6 burndown

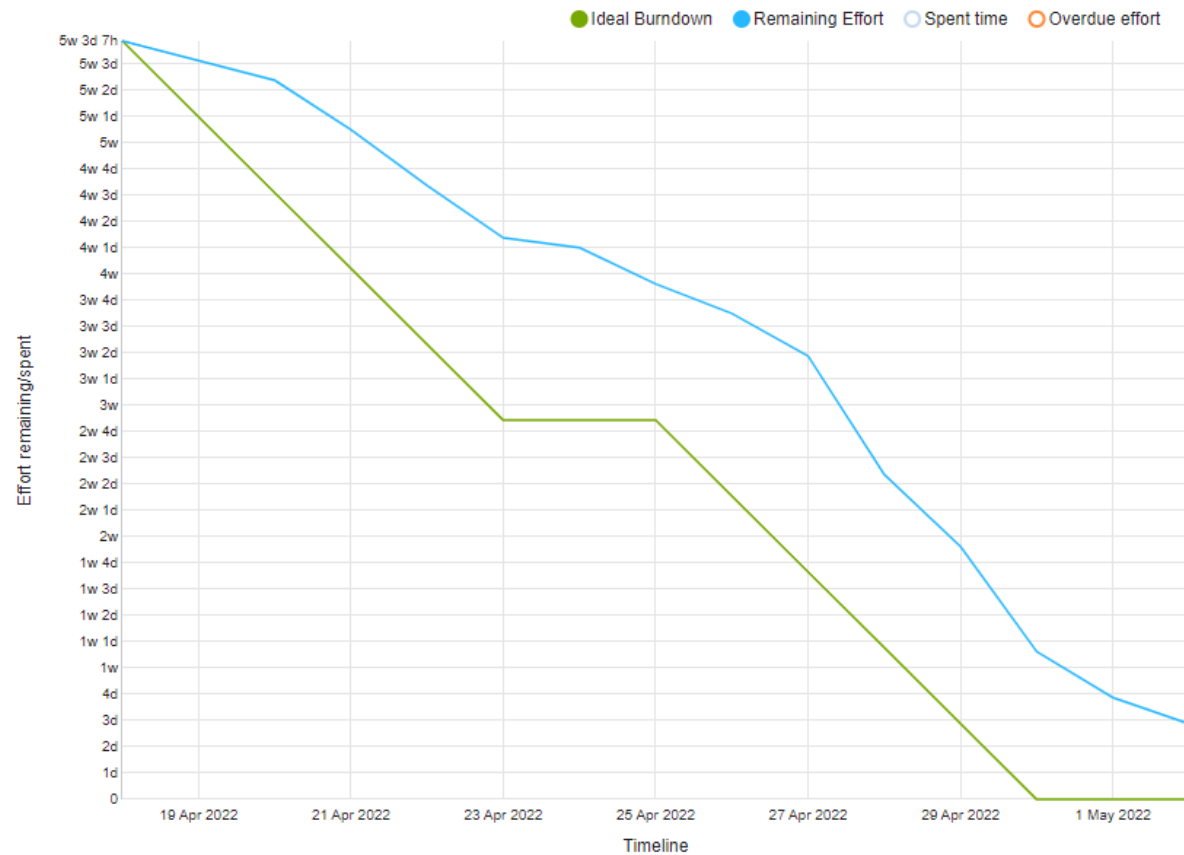


Table 18: Sprint 6 time spent

Team member	Time spent
Finnbogi Jakobsson	89 hours
Gísli Þór Gunnarsson	55 hours
Hákon Hákonarsson	64 hours
<b>Total</b>	<b>208 hours</b>

#### 8.1.7.5 Sprint retrospective

Everything went well during this sprint and no major improvements were needed.

#### 8.1.8 Sprint 7

From the 2nd of May 2022 to 15th of May 2022

In this final sprint the team added a function to detect stacked bar charts, finished creating

unit tests and then implemented a code freeze on the 5<sup>th</sup> of May. During the last week and a half, the team focused on finishing the report and presentation of the project.

#### 8.1.8.1 *Sprint planning*

- What? Finish the project.
- How? By finishing coding and writing the report.
- Who? All Team members.

#### 8.1.8.2 *Sprint backlog*

*Table 19: Sprint 7 backlog*

<b>Sprint backlog item</b>	<b>Prioritization</b>	<b>Estimated hours</b>	<b>Finished</b>
Documenting the code.	A	20	X
Add ability to pick chart type.	B	10	X
Add ability to pick output format.	B	10	X
Setup docker image for frontend.	B	10	X
Host frontend on AWS EB.	B	20	X
Backend returns xlsx file.	C	5	X
Backend returns JSON string.	A	5	X
Write unit tests for image processing.	B	10	X
Write unit tests for text detection.	B	10	X
Write unit tests for colour detection.	B	10	X
Improving code.	A	20	X
Write a risk analysis.	A	20	X
Write a progress report.	A	30	X
Write a system Overview.	A	40	X
Create a user guide.	A	5	X
Crate an operating manual.	A	20	X
Create presentations.	A	30	X

#### 8.1.8.3 Sprint review

The team completed all tasks, and the project is in a good state to hand in.

#### 8.1.8.4 Sprint burndown

Figure 19: Sprint 7 burndown

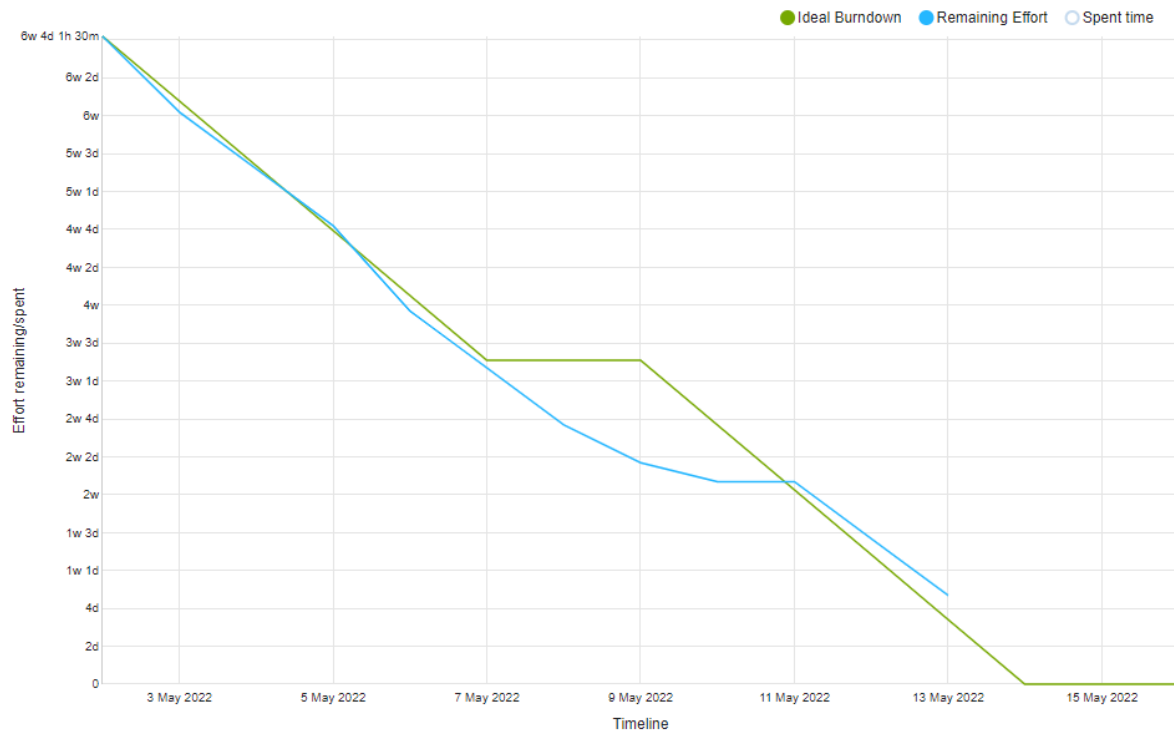


Table 20: Sprint 7 time spent

Team member	Time spent
Finnbogi Jakobsson	91 hours
Gísli Þór Gunnarsson	88 hours
Hákon Hákonarsson	72 hours
Total	251 hours

#### 8.1.8.5 Sprint retrospective

Everything went well during this sprint and no major improvements were needed.

## 8.2 Test images

Figure 20: 12\_entries

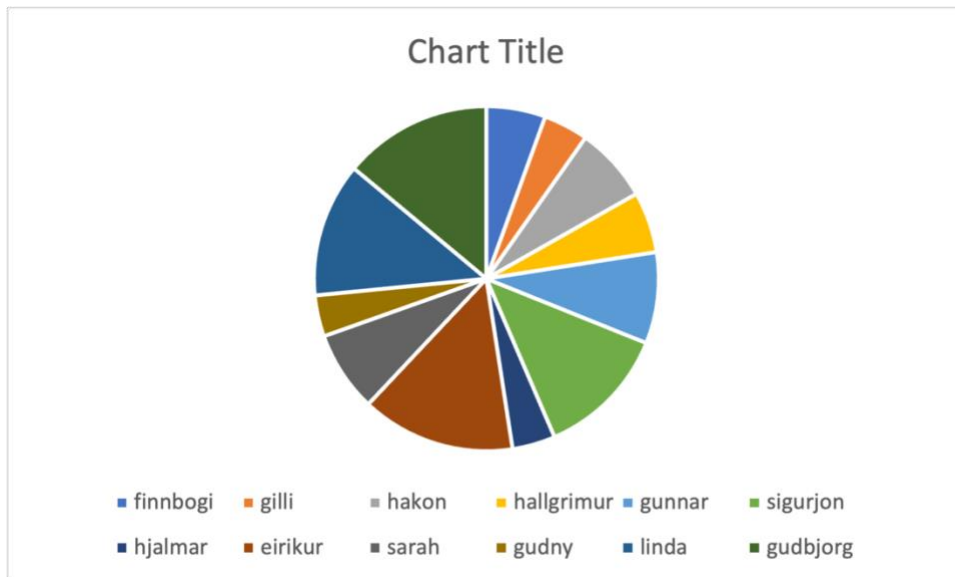


Figure 21: fiveshadesofRED\_20values\_appart

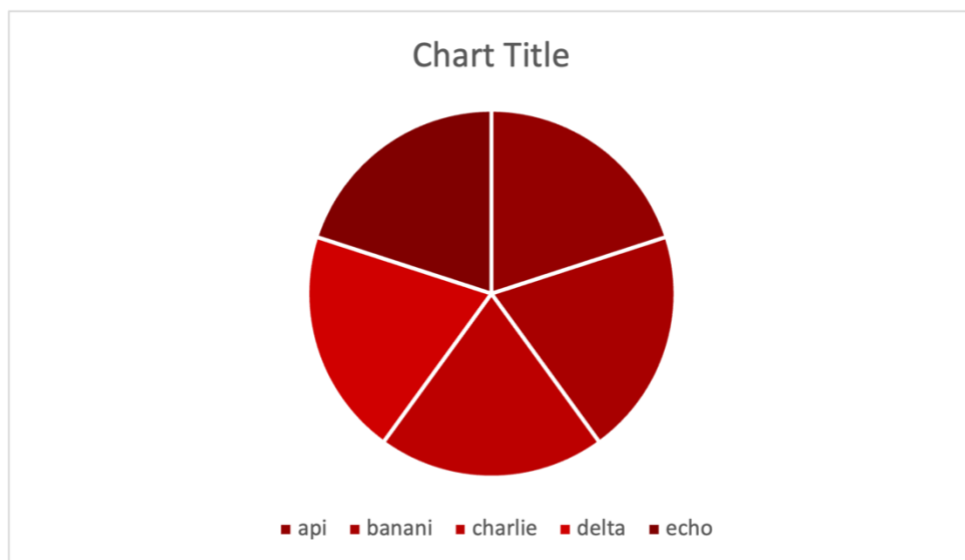




Figure 22: bogatest

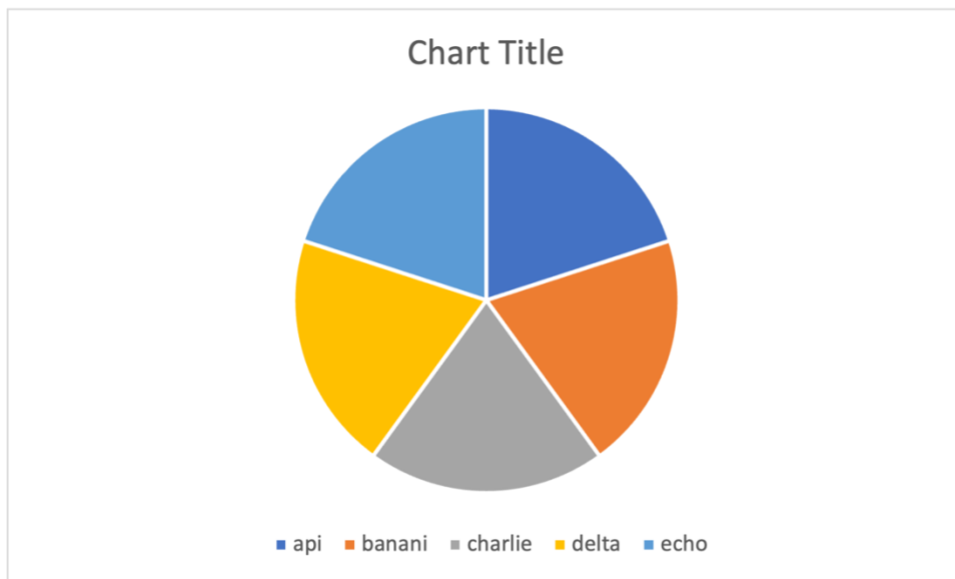


Figure 23: fiveshadesofRED\_10values\_appart

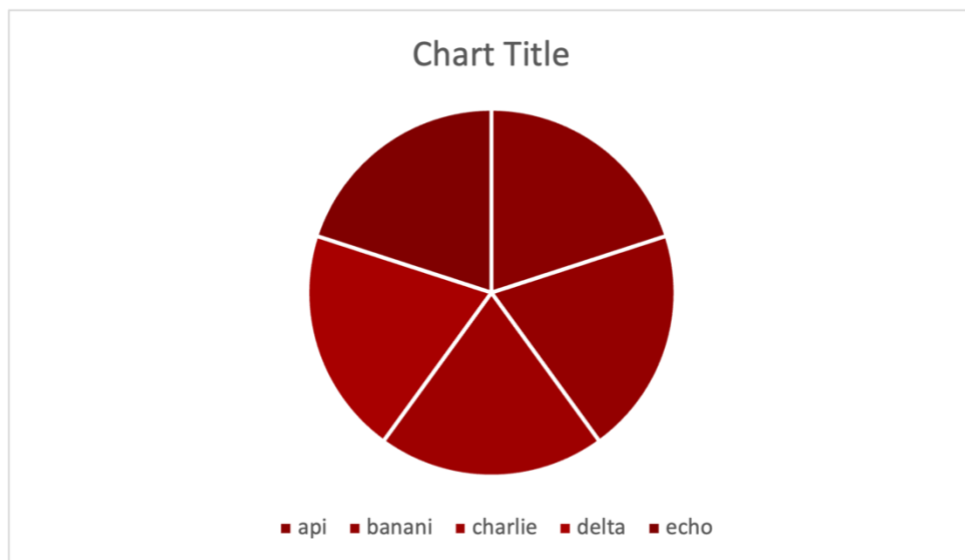


Figure 24: fiveshadesofgray

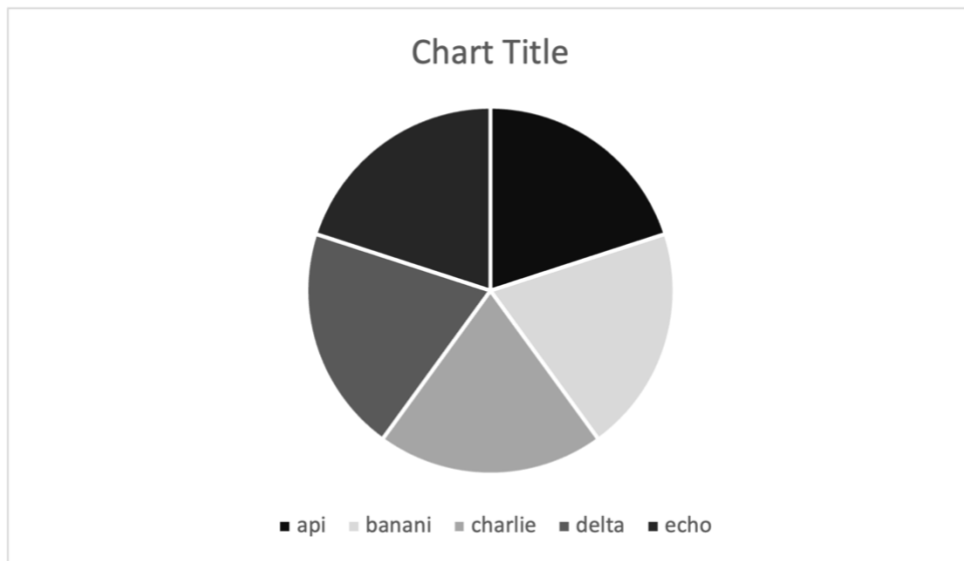


Figure 25: rokk\_stig

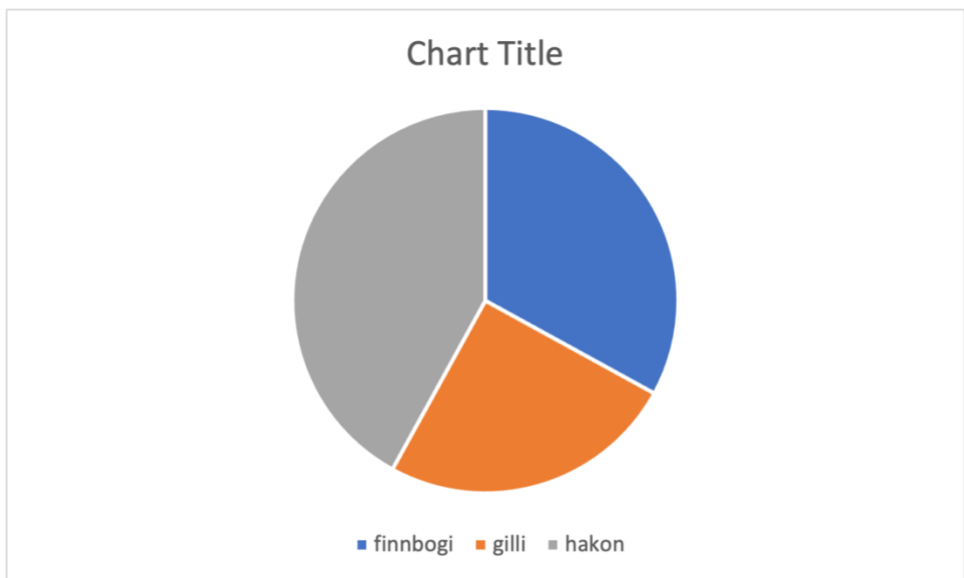


Figure 26: 8\_entries

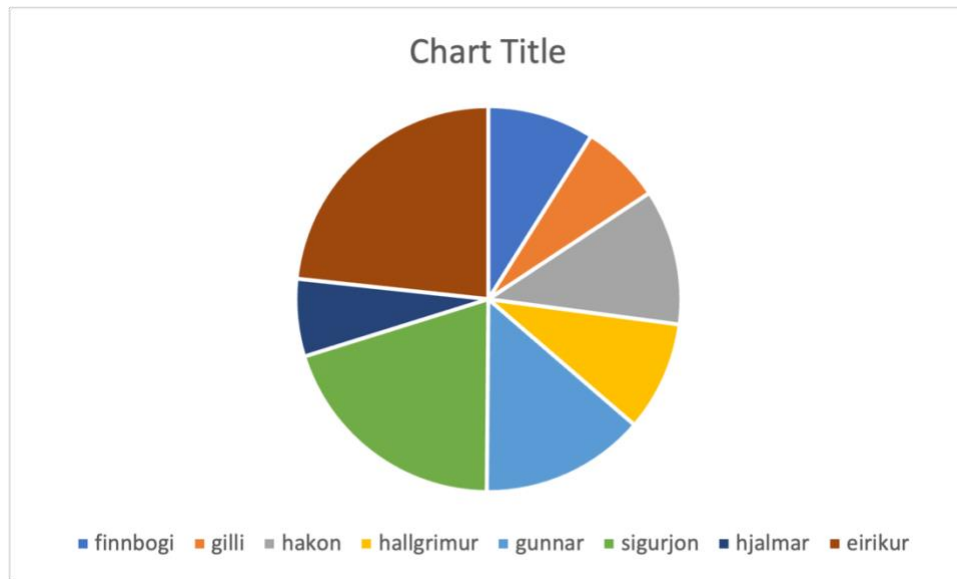


Figure 27: 11\_entries

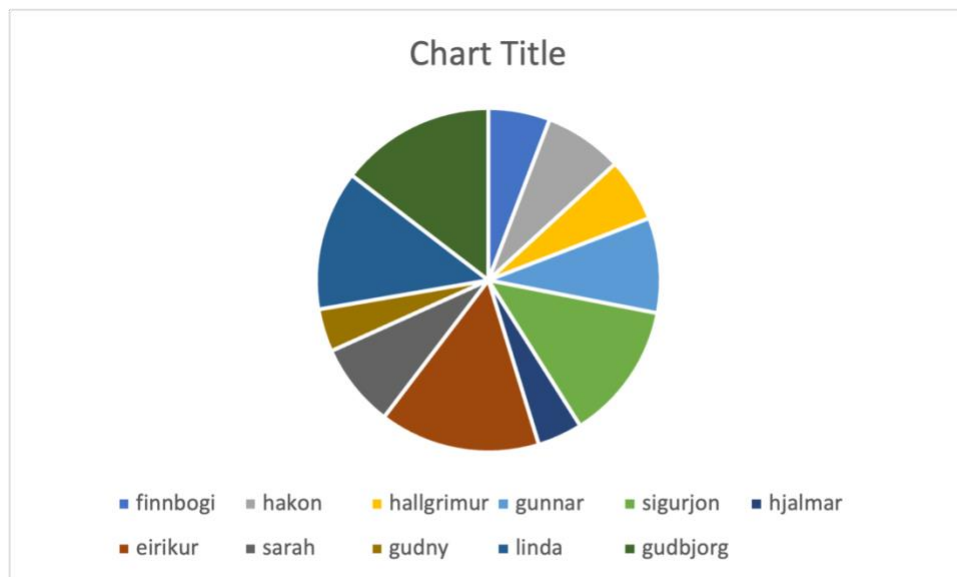
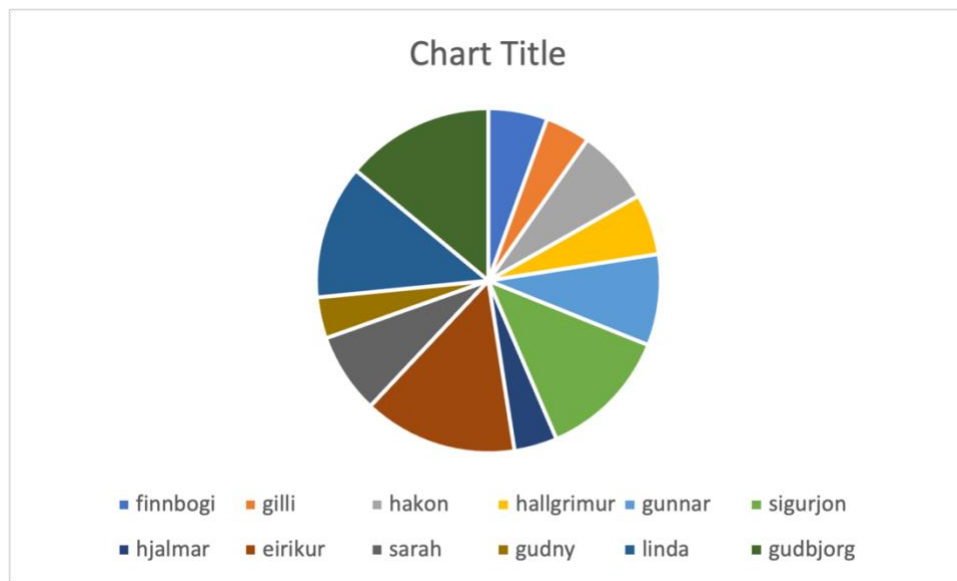


Figure 28: 12\_entries



### 8.3 Test data

Table 21: 12\_entries

<b>finnbogi</b>	<b>33</b>
<b>gilli</b>	25
<b>hakon</b>	42
<b>hallgrimur</b>	34
<b>gunnar</b>	51
<b>sigurjon</b>	74
<b>hjalmar</b>	24
<b>eirikur</b>	86
<b>sarah</b>	45
<b>gudny</b>	23
<b>linda</b>	75
<b>gudbjorg</b>	83

Table 22: fiveshadesofRED\_20values\_appart

<b>api</b>	<b>10</b>
<b>banani</b>	10
<b>charlie</b>	10
<b>delta</b>	10
<b>echo</b>	10

Table 23: bogatest

<b>api</b>	<b>10</b>
<b>banani</b>	10
<b>charlie</b>	10
<b>delta</b>	10
<b>echo</b>	10

Table 24: fiveshadesofRED\_10values\_appart

<b>api</b>	<b>10</b>
<b>banani</b>	10
<b>charlie</b>	10
<b>delta</b>	10
<b>echo</b>	10

Table 25: fiveshadesofgray

<b>api</b>	<b>10</b>
<b>banani</b>	10
<b>charlie</b>	10
<b>delta</b>	10
<b>echo</b>	10

Table 26: rokk\_stig

<b>finnbogi</b>	<b>33</b>
<b>gilli</b>	25
<b>hakon</b>	42

Table 27: 8\_entries

<b>finnbogi</b>	<b>33</b>
<b>gilli</b>	25
<b>hakon</b>	42
<b>hallgrimur</b>	34
<b>gunnar</b>	51
<b>sigurjon</b>	74
<b>hjalmar</b>	24
<b>eirikur</b>	86

Table 28: 11\_entries

<b>finnbogi</b>	<b>33</b>
<b>hakon</b>	42
<b>hallgrimur</b>	34
<b>gunnar</b>	51
<b>sigurjon</b>	74
<b>hjalmar</b>	24
<b>eirikur</b>	86
<b>sarah</b>	45
<b>gudny</b>	23
<b>linda</b>	75
<b>gudbjorg</b>	83

Table 29: 10\_entries

<b>finnbogi</b>	<b>33</b>
<b>hakon</b>	42
<b>hallgrimur</b>	34
<b>sigurjon</b>	74
<b>hjalmar</b>	24
<b>eirikur</b>	86
<b>sarah</b>	45
<b>gudny</b>	23
<b>linda</b>	75
<b>gudbjorg</b>	83

## 8.4 Work distribution by work type

Figure 29: Finnbogi, time spent by work type

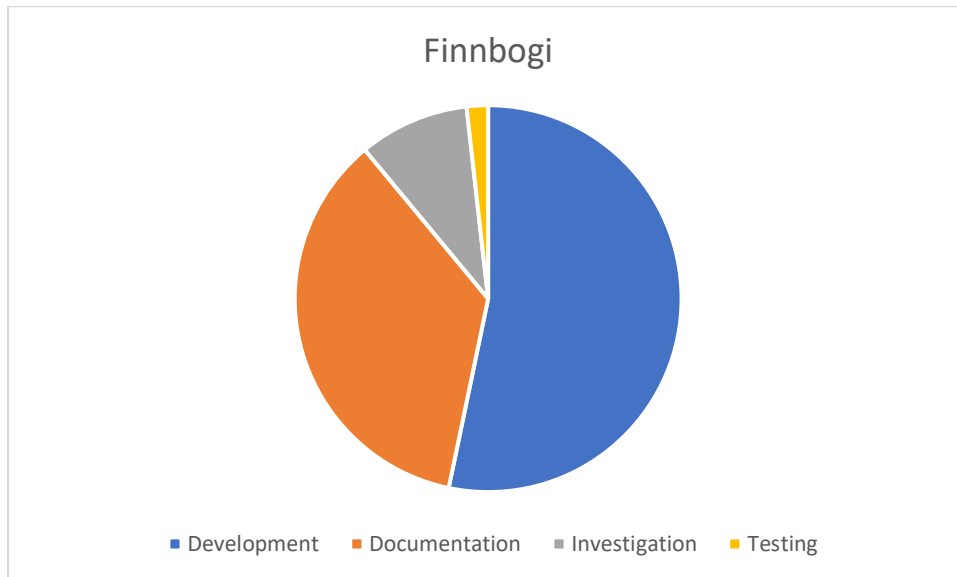


Figure 30: Gísli, time spent by work type

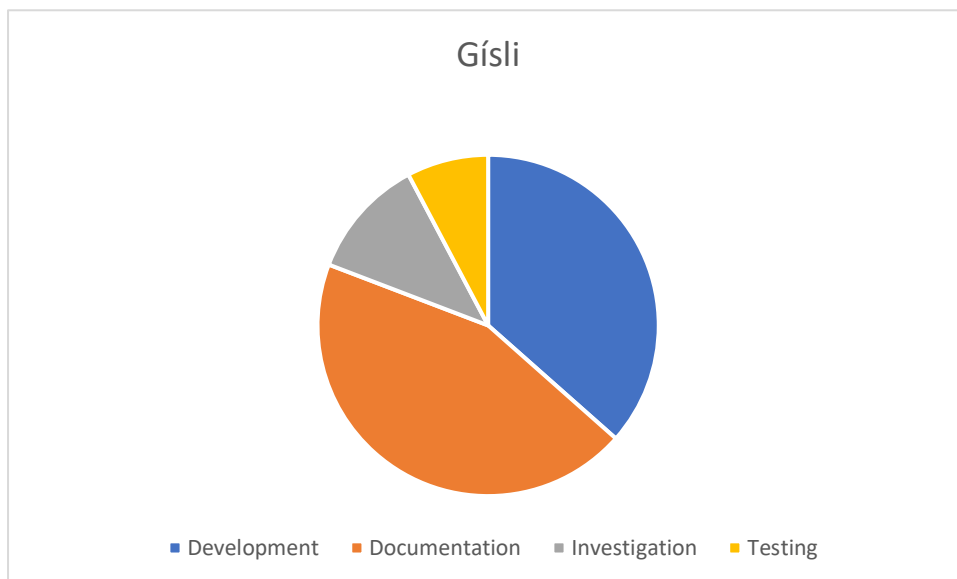


Figure 31: Hákon, time spent by work type

