Graphalytics Global Competition

Clinton Cao
Michiel Doesburg
Sunwei Wang

TU Delft

Graphalytics Global Competition

A Competition Platform to Compare Different Graph-Processing Platforms

by

Clinton Cao
Michiel Doesburg
Sunwei Wang

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

Delft University of Technology,
to be defended publicly on Friday August 25, 2017 at 11:00 AM.

Project duration:	May 29, 2017 – August 4, 2017	
Project committee:	Prof. dr. ir. Alexandru Iosup,	TU Delft, supervisor
	Asst. Prof. dr. Huijuan Wang,	TU Delft, coordinator
	ir. Wing Lung Ngai,	@Large Research Team, supervisor
	Tim Hegeman, B.Sc.	@Large Research Team, supervisor
Client:	@Large Research team	

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

Preface

This thesis presents our work for the Bachelor project, a final project that all computer science students have to finish, before they can acquire their degree of Bachelor of Science. In this project you have a real world client that has a problem and is looking for a solution (a software product) to the said problem. The goal of the project is for the team to come up with a solution, by analyzing the problem and using all the knowledge that they have gathered throughout the bachelor program.

During the first two weeks of the project, we have spent time on analyzing the problem of our client by means of an interview, and coming up with a plan on how we are going to solve the problem. In the remaining eight weeks we have been developing the software product that, we think, serves as a solution to the problem that was proposed by our client.

With this thesis, we try to inform the readers on the research that we have carried out for the problem that was proposed by our client, how we designed our system and how we have implemented our system. We also hope that our thesis can help readers with their own research, for example, whether they are interested in why we have made certain design choices or whether they are interested on how we have analyzed the problem of our client.

Clinton Cao
Michiel Doesburg
Sunwei Wang
Delft, August 2017

Acknowledgements

We would like to thank everyone that has helped us throughout this project. We want to thank Wing for taking time to walk us through the code that he has written for his prototype and also for teaching us how to deploy a website on a server. We have certainly learned a lot from you. We want to thank Tim for his advice on how we can improve our report and thesis. We want to thank both Wing and Tim for taking time to meet with us weekly and providing feedback on our work.

We want to thank our academic coach, Alexandru Iosup, for the following: (1) taking his time to read our orientation report and our thesis, and provide feedback on how we can improve our work, (2) meeting with us to evaluate how the project was going and providing suggestions and advices when we have encountered problems, and (3) for motivating us to achieve the most that we can from this project.

Executive Summary

Since to the rapid development of the graph-processing platforms and techniques the comparison needs to be carried out both systemically and periodically. LDBC Graphalytics is an industrial-grade benchmark, where its goal is to provide a fair and object comparison of graph analysis platforms. Our client, @Large Research Team, is looking for a platform that can used to host global competitions bi-annually. Currently an ad-hoc prototype exist, but it is lacks some important features.

To fully understand the problem of our client, we have conducted an interview with our client. In the interview we have analyzed the different stakeholders that exist within the context of the Graphalytics Global Competition. Based on the analysis that we have done for the problem of our client, we have derived the requirements of the system. The problem analysis can be found Chapter 2.

To see which metrics are used by a competition to give a score to its participants, we have taken a look at different real-world competitions and did a comparison of the competitions. We have also taken a look at different scoring methods and did a comparison of the scoring methods. For the comparison of different competitions and scoring methods, take a look at Chapter 3.

Our approach to the entire project has been the following: we have compared different development methodologies and chosen to use an adapted version of Scrum. This adapted version of Scrum borrows key aspects from Scrum, such as daily face-to-face communication, sprint planning, and sprint reviews, and omit others, such as product owner and scrum master. We have decided to reuse the back-end of the prototype that has been developed by our client and redesign the front-end. We have also analyzed several risks that could hinder the development process and came up with a plan on how to keep the hindrance manageable. Chapter 4 presents our approach.

Our system has two main components: a front-end and a back-end. The front-end serves as the GUI (Graphical User Interface) of our system. The back-end handles all the API calls and the database operations. We went for a single-page design for the front-end. This is to lower the amount of code duplication in our system. The main functionality of the website concerns four schemas: users, benchmarks, competition, and participations. Chapter 5 explains the design of our system. The actual implementation of the system can be found in Chapter 6.

We have used different automated static analysis tools to statically check for defects in our code. We have written tests for both the front-end and the back-end. Though we wrote tests, the coverage of the front-end is not high and there is no information available on the code coverage of the back-end. These are aspects about our project to improve in the future. More about quality assurance is in Chapter 7.

We have encountered several problems during this project, but after discussing with our client we were able to find solutions to our problems. We have learned different things from this project. The lessons that we have learned can be put to use for future projects that are similar to this one. Chapter 8 summarizes the main lessons we have learned.

We have met most of the requirements and answered our research question (and all the sub-questions), but due to the problems that we have encountered there are still issues left on our GitHub repository. These are listed as future work. Chapter 9 discusses our hindsight-look at the project and proposes directions for future work.

Contents

1 Introduction

The increased popularity and size of graphs has lead to the development of different graph-processing platform. Graph-processing platforms are systems that take an arbitrary graph as input and output results. What is outputted as results, depends on what the user has used the platform for e.g. the result of platform X, is the shortest path from point A to point B in graph G. The user has used platform X to find the shortest path within a Graph G.

Because of the wide variety of graph-processing platforms, it is hard to know which platform is best to use for a specific use case. A way to solve this issue, it to benchmark the graph-processing platforms. Benchmarking is used to evaluate the performance of a system. This is done either by, for example, comparing the performance of a system with the performance of a reference system. The performance of the reference system serves as the baseline.

Graph500 is an example benchmark that is used to evaluate graph-processing platforms. Graph500.org ranks the graph-processing platforms based their benchmark score. The score is derived from the amount of edges traversed within a second.

1.1. LDBC Graphalytics

"LDBC Graphalytics is an industrial-grade benchmark for graph analysis platform. The main goal of Graphlytics is to enable the fair and objective comparison of graph-analysis platforms." [9]. In contrast to Graph500, LDBC Graphalytics does not benchmark the platforms on one single algorithm [9]. This benchmark allows the user to compare the different platforms, tune their own platform by locating the bottleneck in their platform and to make their platform future proof [10].

1.2. Problem Description

LDBC Graphalytics needs to provide a platform for vendors to evaluate their results and compare the performance of their graph processing systems. Since the rapid development of the graph-processing platforms and techniques, the comparison needs be carried out both systematically and periodically. A global competition website would be an ideal solution to this problem. Currently there is an ad-hoc prototype, but it still lacks some important features. The prototype uses a scoring system that will later be explained in Chapter 3, but this scoring system is not necessarily the best. The project consists of two main parts:

1. Conceptually designing a sensible scoring system that assigns each participant a fair score that has merit i.e. a vendor can use it as proof of the quality of their system.

2. Designing and implementing a website that can host competitions.

When it comes to the website, we have the options to adapt the prototype into a full-fledged product, or to implement a new system altogether.

1.3. Research & Technical Questions

Because implementing a website to hosts competitions is not a major challenge from a design perspective, our research question focuses on the conceptual design of the scoring system. Our main research question is as follows:

> *How can we create a sensible competition that is fair for all participants, and assigns each participant a score that has merit?*

The main research question can be further broken down into the following subquestions:

1. *(conceptual) What metrics should be used to rank the participants?*
2. *(conceptual) Which possible scoring systems exist and what are their advantages and disadvantages?*
3. *(conceptual) Is it possible to summarize the score from different metrics in a representative single value of merit?*
4. *(technical) What are the main stakeholders and requirements for the website?*
5. *(technical) How to design a system for a global competition of graph-processing platforms that supports the stakeholders and meets the project requirements?*
6. *(technical) How to implement and test the system from point 5?*

1.4. Outline of Thesis

The rest of this thesis is structured as follows: Chapter 2 provides information about the analysis that we have done for the problem of client. Chapter 3 provides the research that we have done on different types of real world competitions and the different scoring methods. In Chapter 4 provides our approach to the entire project. Chapter 5 provides information about the design of our system. Chapter 6 provides information about how we have implemented our system. Chapter 7 discusses what we have done to keep that the quality of our code as high as possible. In Chapter 8, we evaluate how we have worked, as a team, on this project. Chapter 9 provides the conclusion, discussion, and future work.

2

Problem Analysis

In this chapter, Section 2.1 describes the different stakeholders that exist within the context of the competition. In Section 2.2 we list the system requirements, which have been derived from the problem description and discussions with the client. We have prioritized the requirements using the MoSCoW method [26], and agreed upon the results with th project owner.

2.1. Stakeholders Analysis

The different stakeholders that exist within the context of the Graphalytics Global Competition have been derived through discussion with the client. Table 2.1 represents the results in matrix form, displaying the relationship between stakeholders and 'roles'. A role is a specific set of permissions that a user of the Graphalytics website has. Currently there are 4 roles in the website: viewers, submitters, reviewers and admins. a more detailed description of their function is given in Section 2.1.1. A stakeholder can have one or multiple roles listed above.

Roles / Stakeholders	Viewer	Submitter	Reviewer	Admin
System Integrator	✓	✓		
System Developer	✓	✓		
System Buyer	✓			
Hardware Vendors	✓	✓		
Academic Researcher	✓	✓	✓	
Standardization Body	✓		✓	
Project Owner	✓		✓	✓

Table 2.1: Relationship between stakeholders and roles.

2.1.1. Roles

This section provides descriptions for the four different roles that exist for users of the website.

- **Viewers:** This is the default role of the user. Every visitor of the website is automatically a viewer. Viewers do not require authentication, and can see all public results of the competition.

- **Submitters:** They are able to participate in the competitions and upload their benchmark results.
- **Reviewers / Auditors:** These have earlier access to the benchmark results of the company, so they can audit the results. They can be a trusted academic institution, or a company that is specialized in auditing.
- **Administrators:** They can assign roles to users of the website (e.g. grant the role of auditor). They can run and track competitions, and can change any kind of settings the website will have.

Another conceptual role that exists within the Graphalytics Global Competition is that of 'rule-maker'. Rule-makers change the rules of existing competitions or devise new competitions altogether. Administrators will have all the permissions necessary to incorporate changes to competitions or new competitions into the website. As such, rule-makers will not have a separate role within the website. An example of a rule-maker is LDBC, who decide upon the rules of their competition.

2.1.2. Stakeholders

During the stakeholders' analysis, the client mentioned that a conference or an organization which hosts a conference can also be included as a stakeholder, and it will have a specific role as a publisher. The role of publisher would be to release a fixed report about the ranking every so many months. The report will be archived in the website as public data and then be published at the conference. During the meeting, there was no consensus on whether a separate role as publisher is necessary. Since the viewers should be able to view the past years' ranking results from the archive, and admins have the ability to publish results. The stakeholders list are explained in more details below:

- **System Integrators:** Individuals or companies that combine the different subsystems into a whole system. They will ensure the maintainability of the the whole system. An example would be IBM.
- **System Developers:** They can have similar functions to Software Vendors. They are concerned with the software development process of the systems. They can be the companies that sell their software products.
- **System Buyers:** Individuals or companies that are looking for off-the-shelf system solutions.
- **Hardware Vendors:** Companies who wish to sell their hardware. An example would be a company that wants to sell their newly developed processors. If most of the higher ranking systems in the competition use their processors, they can use this as proof of the quality of their hardware.
- **Academic Researchers:** Researchers from trusted academic institutions. Currently, they are the only stakeholders who have both roles of submitters and reviewers
- **Standardization Body:** Organizations who can act as auditors. They will also take part in decision making process of competitions.
- **Project Owners:** The Graphalytics team, they manage the website and competitions.

Table 2.1 shows which stakeholders can assume which roles. Due to the obvious conflict of interest, most stakeholders are not able to assume the role of submitter and reviewer simultaneously. The table represents what is currently envisioned by us and the client. It is likely to change during the project.

2.2. System Requirements

In this section we describe the system requirements of the project. These requirements pertain to the functionality of the product i.e. what it must, should, could and won't do.

must have	viewer	can see the results of published competitions has access to the links to the documentation and repositories relating to the project. can see the website announcement(s). can see the project organization and people involved. can view the published benchmark results for each system. can see the status and other metainformation of the competition. can sign in/sign out.
	submitter	can upload benchmark results of their system(s). can participate in one or more competitions with uploaded results.
	reviewer	can see and audit submissions before the results from a competition are published. can approve or reject a submission.
	admin	can override the decision of the reviewer. can update the status of a competition. can view the status of a submission.
should have	viewer	can sign up using verification via email. can sign up using CAPTCHA. cannot sign in more than x times in y amount of time. can delete their account. can update their profile information.
	submitter	can track the status of their submission. can track the status of their participation. can withdraw from participation in a competition before it is published. can delete a submission if it is not linked to any published competition.
	reviewer	can explain a rejection in a comment.
	admin	can view a user's information. can change the role of a user's account. can restore the system programmatically or manually. can update the website announcement.
	system	automatically performs periodic backups. passwords should be hashed and salted. can update submissions' report format for backwards compatibility in competitions. can verify the syntax and missing fields of a benchmark submission.
could have	viewer	can view the historic results of a system across multiple competitions.
	submitter	can use a submission made for an older version if the data format is a superset of the new version. can register a system as a database object. can link their system to a benchmark.
	admin	can delete a user's account. can delete a user's submissions.
	system	can run our own competition.
won't have	submitter	can declare conflicts of interest with reviewers.

Table 2.2: The requirements of the system. First sorted by the MoSCoW method, then by role within the system. The orange-colored requirements are features only available to registered viewers, as opposed to unregistered viewers.

3

Analysis of State-of-the-Art Competition Platforms

In this chapter we compare several existing competitions and see what kind of characteristics they have. We also compare different kinds of scoring systems in Section 3.2.

3.1. Comparison of Different Competitions

We have looked at several different real-world competitions, ranging from other graph framework competitions to robot competitions, and compared how they ranked their participants. The competitions we have chosen are the most commonly used, often acting as *de facto* standards where the community has reached implicit consensus. In turn, the competitions we have considered are:

- Graph500 [7]. The benchmark used by Graph500 runs BFS on several sizes. The systems are scored on the amount of edges traversed per second. This is measured in GTEPS: giga- traversed edges per second.
- HPCC [8]. The HPCC benchmark runs produces results for four different categories:
  - HPL - *"the Linpack TPP benchmark which measures the floating point rate of execution for solving a linear system of equations."* The performance in this category is measured in Tflop/s.
  - RandomAccess - *"measures the rate of integer random updates of memory."* Measured in GUPS.
  - FFT - *"measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT)."* Measured in Tflop/s.
  - STREAM - *"a simple synthetic benchmark program that measures sustainable memory bandwidth and the corresponding computation rate for simple vector kernel."* Measured in TB/s (TeraByte per second).

  The system's performance value in a category is directly used as its score.
- TPC-C & TPC-H [24]. TPC-C ranks based on categories: tpmC, price/tpmC, and Watts/K-tmpC. TPC-H ranks based on categories: QphH, price/QphH, and Watts/QphH.
- SPECPower SSERT [21]. Result based on ssj operations per watt.
- Top500 [23]. Systems are scored based on Rmax and Rpeak values both measured in Tflop/s.

- SYNTCOMP. This competition has an additional quality ranking, where points are given based on the size of the solution that is given by the participant and by the size of the reference solution [11]. The amount of points received in for the quality metric is proportional to the ratio between the size of the system's solution to the size of the reference solution.
- RoCKIn competition. In RoCKIn each participant is put into a performance class based on its number of achievements for a task. The participants in the class are further ranked by penalizing them on their incorrect behavior while performing a task [1]. A penalization system could be used in Graphalytics, by, for example, adding score to systems based on their performance (low execution time) and subtracting score for high negative factors like energy usage or cost.

	Uses multiple metrics	Uses a separately computed score	Uses penalization
Graph500	No	No	No
HPCC	Yes	No	No
TPC-C & TPC-H	Yes	No	No
SPECPower SSERT	No	No	No
Top500	Yes	No	No
SYNTCOMP	Yes	Yes	No
RoCKIn	Yes	Yes	Yes

Table 3.1: Characteristics of several competitions.

As can be seen in Table 3.1, most competitions go for a relatively simple ranking approach: a system's performance (as measured in Tflop/s or GUPS etc.) is directly used as its score, and multiple categories are kept separate. This has the advantage of being both simple in design and fair. The performance data is directly presented to the user, who can rank the systems according to the metric they consider most important, and draw their own conclusions. The lack of using a separate score in many competing graph-process competition websites seems to indicate that devising an appropriate scoring system is far from trivial. Only two competitions actually use a separately computed score: the quality score and the penalization system in SYNTCOMP and RoCKIn. The quality score in SYNTCOMP comes closest to the idea of a scoring system currently held by us and the client. But the lack of a reference solution for a graph-processing system makes it difficult to implement. In the next section we will discuss several different approaches to scoring. Section 3.2.5 describes a scoring system much alike the quality score, which constructs a reference solution system from the data.

3.2. Comparison of Scoring Methods

The primary challenge in constructing the competition is combining the performance data into a score that makes sense and is fair. If each system was benchmarked on a single algorithm and dataset, this would be trivial. However, the benchmark employs several different algorithm and datasets. What an average use case of a graph looks size (size, interconnectedness, algorithm) is hard to say. This is why we will work under the assumption that each algorithm and each dataset is equally important. That is to say, no weighting will be applied in the scoring system to these factors. Since a system buyer knows the characteristics of their own graph, it might be possible to let a system buyer indicate which metrics are most important for them and adapt the scoring system to value good performance in the chosen metrics higher. But since this is different for every system buyer, this feature would not be a part of the competition process.

To compare different scoring methods, we set up the following experiment. First, we create an input dataset, combining real-world and synthetic data. Then, we analyze several scoring methods using the input dataset. Last, we compare the overall results.

3.2.1. Input Dataset

In Fig. 3.1, we have a table that shows performance data for a particular algorithm for 6 different systems on 6 different datasets. The systems 'EXTRA1' and 'EXTRA2' have been constructed separately to get a better picture of how different scoring systems behave. Although the table actually contains data about execution times, we will work under the assumption that a higher value is better for now. This makes the calculations a little easier, yet does not affect the generality of the method because it is trivial to extend the method for the case where lower values are better or for mixed models. Adapting the scoring system to work when lower values are better requires some kind of inversion somewhere in the calculation, depending on which scoring system is being used.

	WIKI	KGS	CITA	DOTA	DG100C5	GR23
GIRAPH@DAS5	21307	30365	58091	13709	46818	23031
OPENG@DAS5	76	211	11	382	406	1339
OPENG@DAS5AST	63	141	9	245	290	783
PGRAPH@DAS5	2035	2896	4913	1219	4612	2816
EXTRA1	10000	15000	23000	6700	18000	11000
EXTRA2	11000	20000	120000	26000	90000	12000

Figure 3.1: Table with performance data.

3.2.2. Score Based on Distance to Average for each Dataset

The first scoring system we have came up with is using some kind of average to compute a system's score. The idea is to take the average of a column and compute the score for a system based on the difference with that average. For example, the average of the 'WIKI' column is 5870,25. The score for GIRAPH@DAS5 would be 21307 - 5870,25 = 15436,75. Doing this for every column and for every system leads to the score in Table 3.2.

	score minus average	prettified	no negative score
GIRAPH@DAS5	139379	139	139
OPENG@DAS5	-51517	-52	0
OPENG@DAS5AST	-52411	-52	0
PGRAPH@DAS5	-35451	-35	0
EXTRA1	29758	30	30
EXTRA2	225058	225	225

Table 3.2: Table with scores based on average for each column.

It is immediately obvious that using this method results in negative scores. Giving systems negative scores seems conceptually strange. A possible way to deal with this is to simply 'write off' every system which performs below the average and give them zero points instead.

Another apparent problem is the fact that system 'EXTRA2' has a score which is 60% higher than that of system 'GIRAPH@DAS5'. If we look at the original performance table (Fig. 3.1), we see that on three datasets GIRAPH@DAS5 performs twice as well as EXTRA2, and on the other three datasets GIRAPH@DAS5 performs half as well as EXTRA2.

	Pairwise+1	Pairwise+1 squared
GIRAPH@DAS5	27	729
OPENG@DAS5	6	36
OPENG@DAS5AST	0	0
PGRAPH@DAS5	10	100
EXTRA1	18	324
EXTRA2	27	729

Table 3.3: Table with scores based on pairwise comparison. A system receives +1 to its score for a dataset for each system it beats.

Because of this, one could argue these two systems deserve around the same score.

On the other hand, one could argue that since EXTRA2's total score (~290000) is around 50% higher than GIRAPH@DAS5's total score (~190000), the score being around 50% higher as well is fair.

A hidden downside of using an average in this way, is the fact that scores can be negative actually means that comparing positive scores is not meaningful. A 100 is not 50% of 200 if the total range of scores is [-200, 200], but actually 75%. In this case this is still quite easy to figure out, but when the minimum and maximum scores are asymmetrical (e.g. [-430, 250]), which will usually be the case, this quickly becomes very difficult to interpret quickly. Ultimately, using a difference to average leads to a scoring system which is difficult to intuitively understand, and will likely lead viewers to wrong conclusions about the proportionality of two given scores.

3.2.3. Pairwise+1

'Pairwise+1' is how we will refer to the system that is currently implemented in the prototype. Each system is compared to each other system (for each dataset). If its performance is higher, it receives +1 to its score (both systems receive +0.5 if the performance is the same, but this almost never happens). Table 3.3 displays the scores for our example performance data. Contrary to using an average, Pairwise+1 assigns the first and last system equal scores.

Pairwise+1 has a pretty glaring downside however. Looking at the performance data, the second and third system, OPENG@DAS5 and OPENG@DAS5AST respectively, perform about the same. Especially when you consider the huge difference in performance to any system other than these two. However, OPENG@DAS5 has received a score of 6 and OPENG@DAS5AST a score of 0. The problem is obvious: pairwise+1 does not in any way take into account by how much a system beats another. This is also clear when looking at the scores for GIRAPH@DAS5 and PGRAPH@DAS5. In every dataset, GIRAPH@DAS5 performs roughly ten times better than PGRAPH@DAS5. Since this is true for every dataset, you could say GIRAPH@DAS5 performs around ten times better in general. Yet PGRAPH@DAS5 has a score of 10 to GIRAPH@DAS5's score of 27. This is 37%, nearly four times higher than the supposed 10% PGRAPH@DAS5 performs at.

A technique to mitigate this is to square the final score. Table 3.3 displays this. Admittedly, this does nothing to incorporate the factor of by how much a system beats another. Instead, it works on the assumption that beating a number of other systems is not a 'linear' accomplishment. This implicitly assumes that beating a high amount of systems also means they are beaten by a lot. A system beating twice as many systems as another means that system is far better than just twice as good. As we can see, GIRAPH@DAS5 now has a much higher score (more than 7 times as much) than PGRAPH@DAS5, and is now much closer to what we would call fair. OPENG@DAS5 still has 'much' more points than OPENG@DAS5AST, despite being only marginally better. But the amount of points OPENG@DAS5 has is, relatively, much lower now compared to the other systems. This works quite

	Scaled
GIRAPH@DAS5	4,52
OPENG@DAS5	0,03
OPENG@DAS5AST	0
PGRAPH@DAS5	0,4
EXTRA1	2,06
EXTRA2	4,67

Table 3.4: Table with scores based on directly scaling a systems performance value.

well for the current example data. However, consider the case when every system performs roughly equally, but one system beats every other system by a slight margin. This system will receive a huge score despite being only marginally better. Applying tricks like squaring the score can help make the scores fairer in specific scenarios, but will worsen them in others. Applying tricks like squaring the score can help. Considering the fact that Pairwise+1 does not take into account by how much a system beats another, and that any technique to make the scoring fairer are mostly band-aid solutions that even lead to increased unfairness in some cases, makes Pairwise+1 look like an unattractive option.

3.2.4. Scaling to [0, 1] per Dataset.

Another way to assign scores is directly scaling a system's performace value to a value in [0, 1]. For a dataset, the best performing system corresponds to 1 and the worst corresponding system corresponds to 0. This method has several benefits compared to the aforementioned methods. Contrary to pairwise+1, it takes into account by how much a system beats another, and awards it points directly based on that value. As you can see in Table 3.4, OPENG@DAS5 has received an only marginally better score than OPENG@DAS5AST's score of 0. This corresponds well to the relative performance of these two systems. PGRAPH@DAS5 has a score of 0,4, roughly 10% of GIRAPH@DAS5's score of 4,52. This corresponds well to PGRAPH@DAS5's performance which is roughly 10% of GIRAPH@DAS5's.

Contrary to using an average, there are no negative scores. Moreover, GIRAPH@DAS5 and EXTRA2 have roughly equal scores, and seeing as to how GIRAPH@DAS5 beats EXTRA2 by twice as much in half the datasets and vice versa for the other half, this seems fair.

3.2.5. Fractional Difference Method

The fractional difference method works similarly to the 'Scaling to [0, 1]' method. Where [0,1] scaling takes as the best case the best performing system for a dataset, the fractional difference method computes a theoretical global best performing system. This theoretical best system combines only the best scores of all participating systems for each algorithm and dataset. Scoring is then based on how close a system's performance is to the theoretical best system. This is computed as a fraction. The fractions are then summed to arrive at a score. Since our example data is only a single table, this method would arrive at the same score as [0,1] scaling. We will have to see how the fractional difference method performs in practice to draw conclusions on its merit.

3.2.6. Analysis Across Methods

The first scoring method could lead viewers to wrong conclusions, therefore it is not a good method to use. Both Pairwise+1 and Pairwise+1 squared does not does not take into account how much a system beats another. This might be an issue when the performance of the systems does not differ by much. System A that was 1 millisecond faster than system B, received 1 point and the other 0. This might not be fair to the participants. The remaining two methods does take into account of how much a system beats another. Using one of these two methods might be fairer to the participants, in comparison to Pairwise+1 and PairWise+1 squared.

4

Approach to the Entire Project

This chapter describes the approach to our project. First, we are going to illustrate the development methodology which is discussed in Section 4.1. Then, in Section 4.2, we discuss whether we will be reusing components from the current prototype. Finally, in Section 4.4, we analyze the risks of this project.

4.1. Development Methodology

There are several options for the development methodologies available for us, such as Scrum methodology, Waterfall methodology and XP (Extreme Programming). For the software development process, after weighing the advantages and disadvantages, we chose to work according to the agile methodology in an iterative development process. Considering the relatively short development period of eight weeks, we worked with weekly sprints. Weekly meetings with the client will be held to receive feedback on performed work and ensure the project continues in the right direction.

From working together in previous projects we all know each other well and are comfortable picking and choosing the elements of Agile and Scrum that work for us. Managing a project with three team members is relatively easy, so we try not to get bogged down in administrative and managerial meta-work. We borrow some key aspects from Scrum like daily face-to-face communication, sprint planning, and sprint reviews. While other aspects such as product owner and scrum master are not employed as we consider them unnecessary and a waste of time. From our past experience, it is more efficient for our entire team to fulfill the duty of those two roles than having someone specific to do it.

Our team has opted out the choice of waterfall methodology. Since it requires the complete understanding of requirements up front, and there can be no change in the requirements later in the stages. However, we would like to work continuously with our clients and have a flexible development process. Thus waterfall is not a viable development methodology for us.

There are mainly two reasons why we chose to use Scrum over Extreme Programming. Firstly, Extreme Programming is more suitable for complicated projects since it takes more time and human resources with Pair Programming practice [9]. Our team would prefer to work individually on assigned tasks and review each other's work. Secondly, our team are have more previous experience with Scrum and it proved to be working very well for our teamwork.

4.2. Adaptation of Current Prototype

Currently a prototype exists that has been put together by our client. This prototype includes both a back-end and front-end. It is up to us to decide how much of the prototype we will use in the implementation of the system: from directly building on top of the unchanged prototype to adapting

parts to not using it at all. After looking at the code and deliberating amongst each other, we have decided the following: (1) We will use the current back-end as is, possibly refactoring or restructuring it slightly, and extend it with additional features. (2) We will port the front-end JavaScript code to TypeScript, redesigning and restructuring the architecture in the process. For the back-end in prototype, our client has designed it to be a single file. Our new design has separated the back-end code into 6 main folders: benchmark, competition, participation, project, update, users according to their functions, it is easier to maintain and rework. We have made this decision because of the following reasons:

- We have more experience with front-end web development than with back-end development. We think that finding and understanding a new back-end framework will take us more time than understanding how the current back-end is put together and adapting it from what it is now. Furthermore, the current back-end is in relatively good shape, so there is no reason to believe we could do better with different technology.

- The front-end is in relatively bad shape. The current front-end consists of a collection of large files written in JavaScript. We expect that extending the current front-end is going to lead to very 'hacky' coding; every new feature becoming more difficult to add than the last. We fear the project will end up as a mountain of incomprehensible code which will be hard to maintain and even harder to extend. Comprehensibility, extendibility, understandability, documentation, testing are all aspects of code quality that we value greatly. Each of which will be hard to ensure using the current front-end code as starting point.

4.3. Project Planning

The main timeline of this project can be summarized in a Gannt Chart in Figure 4.1. There are in total of ten weeks listed horizontally, and eight major tasks listed vertically. In the first two weeks, it was the research phase, the team focused on meeting our academic coach and our client, requirement analysis and stakeholder analysis were conducted, and orientation report was delivered at the end of research phase. Week three to four was mainly setting up the front-end architecture and integrate the existing code from back-end. Week five through seven consists of implementing the must-have and should-have features of front-end plus back-end. Week eight through nine was used to fine-tune the system and implementing unfinished features. Week ten was used mainly for writing the report.

4.4. Risk Analysis

We have analyzed several risks that we think could hinder the development process. These risks must be addressed to keep the hindrance to a minimum. The risks that we have analyzed are explained in the following subsections.

4.4.1. Web Development

We think that the most challenging part for us, is the back-end development. This is because we have little experience working on back-ends. This can cause hindrance to the development process as we will likely have to spend much time learning and searching for solutions. Therefore for a task where a new feature will be implemented in the back-end, we will overestimate the amount of time that is needed for the task.

4.4.2. Growing List of Desired Features

During the course of this project, the list of desired features might keep growing, due to the feedback that we will be receiving from our clients or due to additional features that our client would want to have in the product. The reasons for this are that Graphalytics is still a on-going project and

the technical specifications of Graphalytics are still being finalized. There might be additional features added during the process of development. Another reason can be the multiple stakeholders that exist for this project. Their input to the large research team might present additional desired features.

This could lead to a list of features that is too large for us to finish within the time frame of this project. Therefore before adding a new feature to the list, we will discuss the priority of the feature with the client. With the help of a discussion, we can maintain a smaller list where everything will be implemented within the time frame of the project.

4.4.3. Personnel Availability

A team member has informed the team, our coach and our client that he will be in China between July 8th and August 17th, and he will be working remotely during this period. Due to the different time zones, the time difference between the Netherlands and China is six hours, so 10:00 A.M in the Netherlands is 16:00 P.M in China. Communication will be a challenge. It is therefore important for us to keep each other updated: all team members will be informed about the new changes as soon as possible. We will communicate mainly through Telegram and WhatsApp. The Slack channel is also an alternative for keeping each other updated. Furthermore, the issues and pull requests on Github/ Gitlab are other ways of communication, they will be used together with Telegram and Slack to keep each other updated. Skype call is another solution. Due to the six hours of time difference, the meetings will be planned ahead through Slack or Telegram.

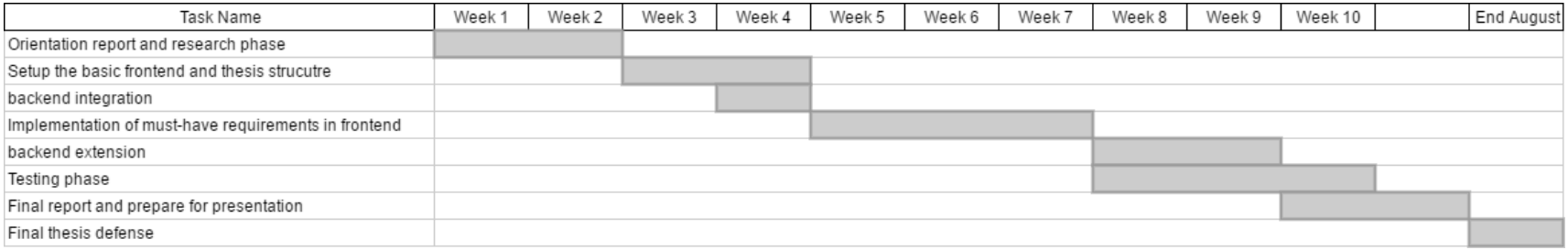


Figure 4.1: Visual Plan represents in a Gantt Chart.

5

System Design

This chapter focuses on the design of the our system. First, in Section 5.1, we present the two main components of our system. Then, in Section 5.2, we analyze how we have designed the front-end of our system. Section 5.3 introduces the design of the different views for the different roles. Section 5.4 proposes the implementation of the life-cycles of the database objects.

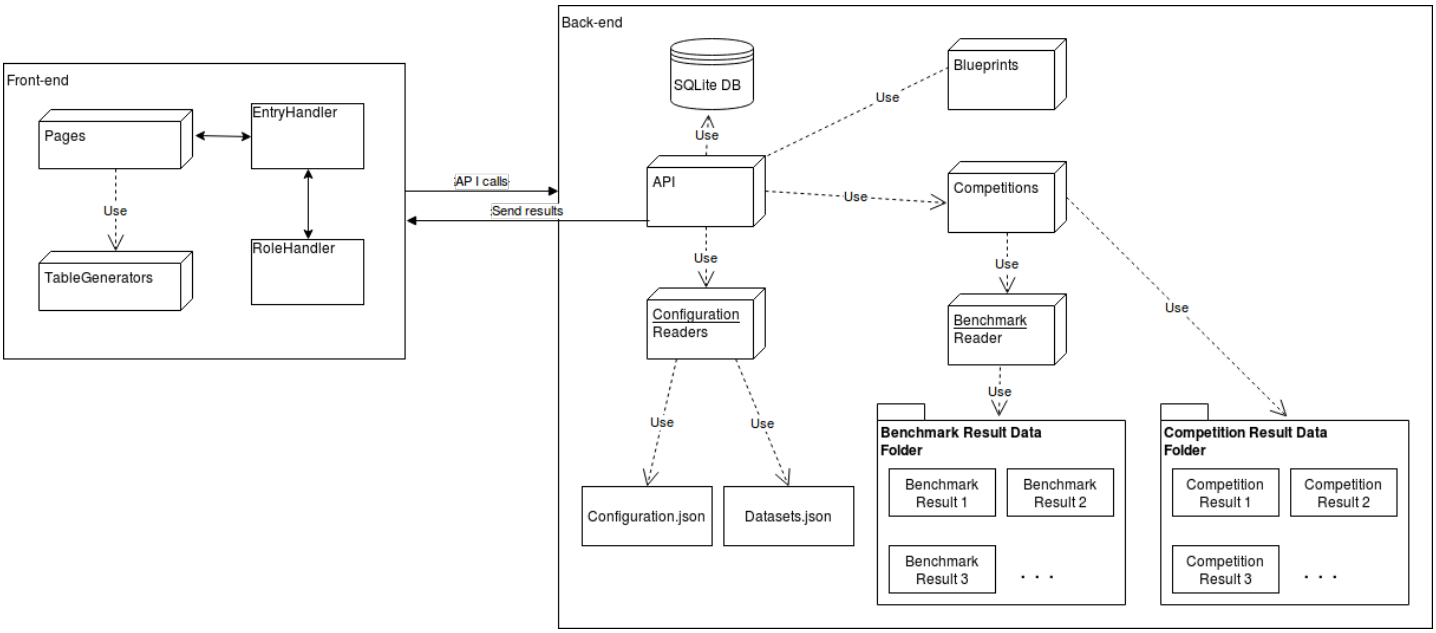


Figure 5.1: Global overview of the website.

5.1. Main Components

The two main components of our system are the following:

1. Front-end: this component is responsible for rendering the different pages of the website. It serves as the GUI of our system.
2. Back-end: this component is responsible for handling the different API calls and how data is stored in the database.

Fig. 5.1 provides a global overview of the two components. We have decided to use two main components for the following reasons:

- It is easier to debug the system: if there is only one single main component and there is a bug in the code, it will take a longer time to identify where the bug is located. With two main components, the developer can narrow it down in which main component the bug is located and then keep searching within that component.
- It makes the development of each main component easier for the developer e.g. if there is a critical bug that causes the front-end to crash, then the developers whom are working on the back-end do not have worry about the bug. But if there is only one single component, the developer will first have to fix the bug (or wait for another developer to fix the bug) before they can continue with their work.

5.2. Single-page Design

We have chosen for a single-page design for the front-end due to the following reason: using the multiple-page design could cause code duplication. If there are pages that are almost identical, then there is duplicated code. Using a single-page design, code that renders the elements that are the same for all pages can be generalized. The developer would only have to write code to render the elements that are unique for a particular page.

5.3. Role-specific Views

As mentioned in Section 2.1.1, there are four different roles which exist for users of the website. The front-end should therefore show the right contents to the users, based on their role e.g. submitters should not be able to see the contents that are only available to administrators. The different views is shown in Fig. 5.2. We tried to satisfy the requirements (where a role involved) that is listed in 2.2 with our design of the different views.

The tabs that can be seen for the role of a viewer are the same for all other roles. The "Account" tab is the same for all registered users (admins, users and reviewers). The "Benchmark" and "Participation" tabs are the same for both user and reviewer. A further explanation of the tabs for each view is given in the following:

For the Viewer View:

- Overview: this is where the viewer can go and view the main page of the website.
- Competition: this is where the viewer can see all competitions and their results (if published).
- Documentation: this is where the viewer can find all the technical documents, slide presentations and academic publications
- Repository: this is where the viewer can find the repository for the LDBC Graphalytics benchmark software.
- Dataset: this is where the viewer can the dataset that are used for the benchmarks.
- About: this is where the viewer can find the information the organizations and people that are involved with the project.

For the User View:

- Account: this is where the user can find information of their account.
- Benchmark: this where the user can upload their benchmark results.
- Participation: this where the user can participate in a competition.

For the Reviewer View:

- Review Participation: this is where the reviewer can review the participation of a user.

For the Admin View:

- Manage Competitions: this is where an admin can open or close a competition. This is also where the status of the competition can be changed.
- Manage Participations: this is where an admin can see and change the status of the participation of a user.
- Assign Reviewer: this is where an admin can assign a reviewer to review a particular participation of a user.
- Manage Users: this is where an admin can see all the delete a user or change the role of a user.
- Manage Submissions: this is where the an admin can view and manage all the benchmarks submissions.
- Manage Announcements: this is where an admin can manage all the announcement on the website.

5.4. Database Object Life-cycles

In this section we describe the life-cycle of benchmark, competition, and participation objects in the database. The life-cycle is depicted by a state diagram showing which states an object can be in and which transitions are possible.

5.4.1. Benchmark Submission Life-cycle

The result file produced by running the benchmark is referred to as a benchmark submission. Figure 5.3 depicts the state diagram which describes the benchmark submission's life-cycle:

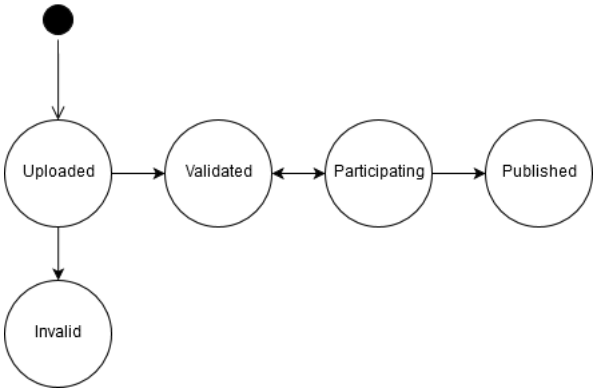


Figure 5.3: The life-cycle of a benchmark submission.

Figure 5.2: Role-specific content.

The benchmark submission has the following states:

- Uploaded: When a user submits a benchmark result file the life-cycle begins. Its state is subsequently 'Uploaded'.
- Validated: An automatic validation process is run which checks if the uploaded file is a valid benchmark result file. The file should be a valid JSON file according to the benchmark report format, and the file should not be too large.
- Invalid: If the automatic validation process fails, the state becomes 'Invalid' and the benchmark can not be used.
- Participating: As soon as a user uses this submission to participate in a competition, its state will be 'Participating'. If the user withdraws the participation, the state will revert to "Validated".
- Published: When the competition the benchmark submission is participating in is published, its state becomes 'Published'. The benchmark will be available to be viewed by any visitor of the website. The user can not unpublish their benchmark submission.

5.4.2. Participation Life-cycle

A 'participation' is a combination of a benchmark submission which participates in a competition. Its life-cycle is described by the state diagram in figure 5.4. A participation has the following states:

- Submitted: When a user selects one of their benchmark submissions to participate in a competition, a participation's life-cycle begins. At this point the benchmark submission needs to be reviewed, before it will be included in the computation of the competition results.
- Accepted: When a reviewer approves the benchmark submission, the status of the participation becomes 'Accepted'.
- Rejected: Similarly, when a reviewer rejects the benchmark submission, the status becomes 'Rejected'.
- Withdrawn: The user can choose to withdraw their participation both before and after review, and regardless of the outcome of the review. If the competition the benchmark submission participates in has been published, the user is no longer able to withdraw the participation.

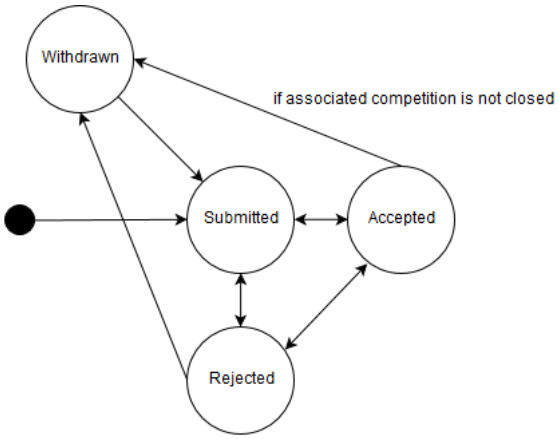


Figure 5.4: The life-cycle of a participation.

5.4.3. Competition Life-cycle

Competitions are created and managed by admins of the website. Figure 5.5 is the state diagram which describes the life-cycle of a competition. Competitions have the following states:

- Announced: When an admin creates a competition, the life cycle of the competition begins. Users and visitors of the website can see the competition, but not participate in it.
- Opened: After an admin has opened the competition, users can participate in the competition with their benchmark submissions.
- Closed: An admin can close the competition after the users participated in it. It can be re-opened by the admin as well.
- Published: When an admin decides that a competition results are ready, admin can publish the selected competition. Only after the competition is published, the visitors and users will be able to view the results of the competition. However, if an admin published a competition by accident, the admin can still unpublish it, and the competition will goes back to "Closed" state.

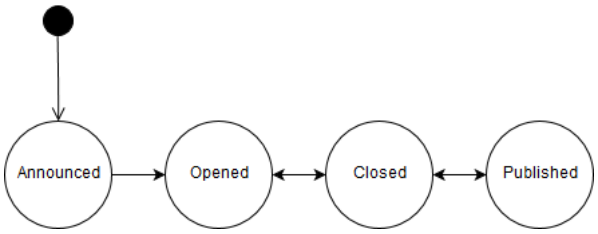


Figure 5.5: The life-cycle of a competition.

6 System Implementation

This chapter focuses on the implementation of our system. First in Section 6.1, we give information on the technologies that we have used to develop the system. In Section 6.2 and 6.3, we discuss the front-end architecture and back-end architecture, respectively. Then, in Section 6.3.1, we give an overview of the back-en. Finally, in Section 6.4, we discuss the security of our system.

6.1. Technology Used

6.1.1. Programming Languages

For front-end development, we used TypeScript, HTML, and CSS. We choose TypeScript over JavaScript, because we strive for modularity and prefer working with OOP patterns. *"Typescript is typed superset of JavaScript that compiles to plain JavaScript" [13].* As TypeScript is transpiled to JavaScript, it works on any browser and operating system. We used Python and Flask (a Python framework that is used for the development of web applications [18]) for back-end development because the back-end is already written using this technology.

6.1.2. Frameworks & Tools

In this section we provide a list of framework and tools that we have used to develop our prototype. A little bit of information is given for each tool/framework and the reason that why we chose to utilize the tool/framework.

Node.js

"Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world" [15]. We used Node.js to compile the scripts that are written in TypeScript.

Integrated Development Environment (IDE)

In order to make sure that each team member works with the same setup, we decided to use JetBrain PyCharm as our IDE. By working with the same setup we avoid running into compatibility issues with libraries and IDE e.g. library *X* is supported by IDE *A* but not by IDE *B*.

Bootstrap

Bootstrap is a framework that is used for easily creating lay-outs in websites [3]. Besides Bootstrap there are other framework that are used for front-end web development. A few examples are: Foundation, Zimit, Ink, HTML Kickstart and Kickstrap. We chose to use Bootstrap for the front-end development because we have more experience with Bootstrap than with the other front-end frameworks.

Flask

As mentioned in Section 4.2, we do not have a lot of experience with back-end development and we will be reusing the current back-end. And as mentioned in Section 6.1.1, the back-end is written in Flask. Therefore, we used Flask for the development of the back-end.

Flask-Mail

One of the most basic functions in a web application is the ability to send emails to your users.[17]. Flask-Mail is used to send a confirmation e-mail when a new user has just registered on the website.

Flask-Bcrypt

Flask-Bcrypt is a Flask extension that provides bcrypt hashing utilities for your application [19]. We used Flask-Bcrypt to hash and salt the passwords of the users.

Flask-SQLAlchemy

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. [19]. We used Flask-SQLAlchemy to handle all the database operations.

6.2. Front-end Architecture

The essential backbone of the front-end is Pages module. The different competition page, benchmark page, user page, etc. are all in the Pages module. There are various tables in these pages displaying e.g. competitions' information, users' accounts information. They are generated using TableGenerators module. The user of the websites can be categorized in three roles: participant, reviewer and admin. Each of them can see different content of the website, and this is controlled by EntryHandler and RoleHandler class.

6.2.1. Page Module

Page module contains the all the classes for different pages on the websites. PageGenerator is a special class in this module, it coordinates the generation of the contents of all the pages. Page class is the superclass in the Page module, all other classes except PageGenerator are the subclasses of Page class, such as: OverviewPage, UserAccountPage, AdminAssignReviewerPage, etc. See Page Module architecture in Fig. 6.1 .

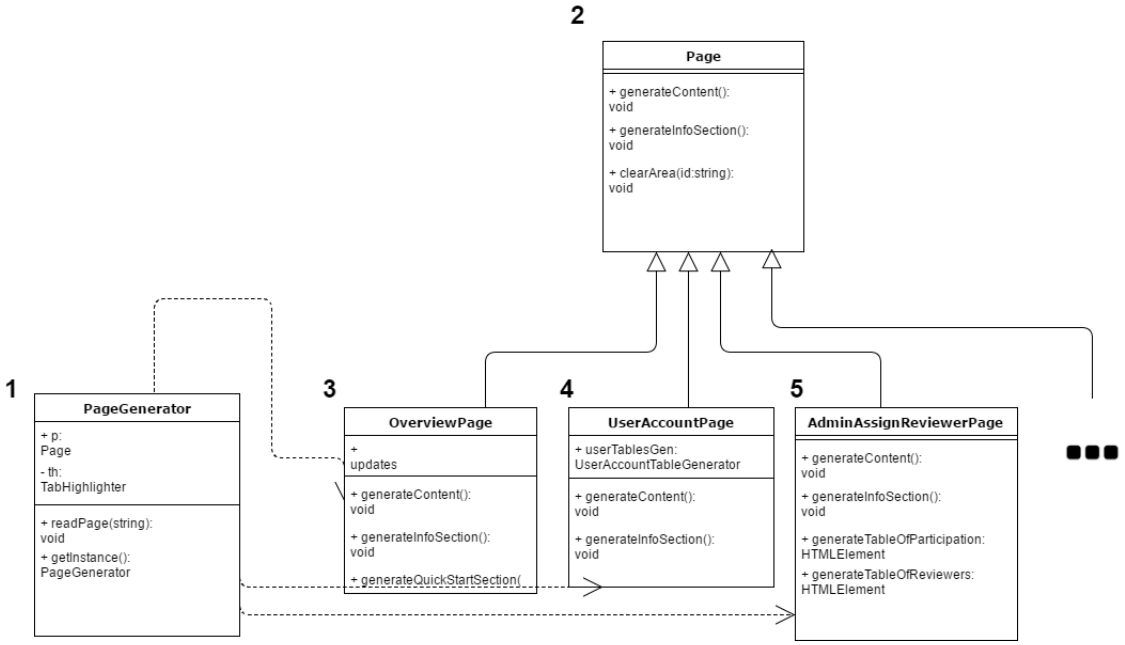


Figure 6.1: Class diagram of the Page module.

1. The function of the PageGenerator class is to generate different page content based on the different tabs on the sidebar of the page. It has a main method: *readPage*, which makes sure that all the pages generate the correct content and highlight the right tab.

2. Page class is the superclass of other different types of page classes. It has the structure of all the subclasses.

3. OverviewPage class represents the Overview Page, it has a brief introduction of the Graphalytics benchmark project, and also has section of how the website works overall.

4. UserAccountPage class displays the account information in the view of the user. It displays the information of the the user that is currently logged in, such as: username, email and organization.

5. AdminAssignReviewerPage class generate the content of the AssignReviewer Page, but it is only available for admins. This class has two tables, one for the reviewers, and there is dropdown list of participations that can be assigned to the reviewers. Another table is for the participations, in this table, it can be seen whether the participation has been assigned to a reviewer.

6.2.2. TableGenerator Module

The most important class in the TableGenerator module is the TableGenerator class, it is an abstract class for other TableGenerator classes e.g. AdminAccountTableGenerator class, BenchmarkTableGenerator. See TableGenerator module architecture in Fig. 6.2

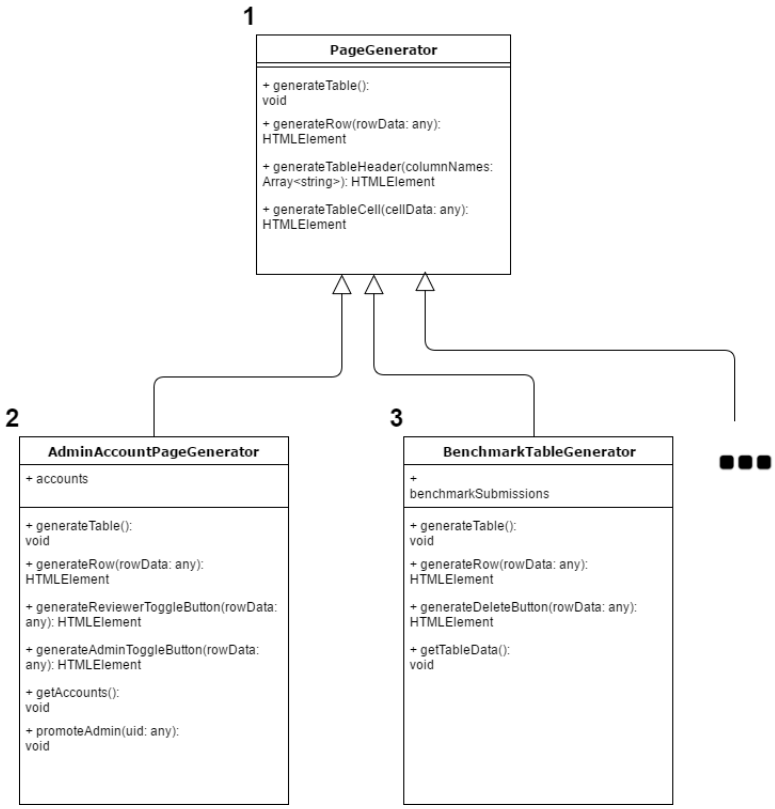


Figure 6.2: Class diagram of the TableGenerator module.

- TableGenerator is an abstract class for all the TableGenerator classes. This class contains the abstract methods such as *generateTable*, *generateRow* etc. They are implemented accordingly in the subclasses of TableGenerator. TableGenerator Class also contains non-abstract methods such as *generateTableHeader* and *generateTableCell*. These are utility methods, which are used in all the subclasses of TableGenerator.

- AdminAccountTableGenerator class extends from TableGenerator class, the methods like *generateTable* and *generateRow* are implemented, and override the abstract methods in TableGenerator.There are also concrete methods like *generateReviewerToggleButton* and *generateAdminToggleButton*. The AdminAccountTableGenerator class generate the tables and buttons for admins to manage the users of the website.

- BenchmarkTableGenerator class extends from the TableGenerator class as well. Similar to AdminAccountTableGenerator class, it contains concrete methods which override the abstract methods in TableGenerator class.

6.2.3. EntryHandler Class

EntryHandler class has several important features:

- Login feature for the user.
- Sign-up feature for the user.
- Log-out feature for the user.
- Creating account panel (drop-down list for different roles of the user).

6.2.4. RoleHandler Class

The main function of the RoleHandler class is to handling different roles of the user of website. It has several features:

- set the different roles of the user.
- generate the view for the visitor.
- generate the view for the user.
- generate the view for the reviewer.

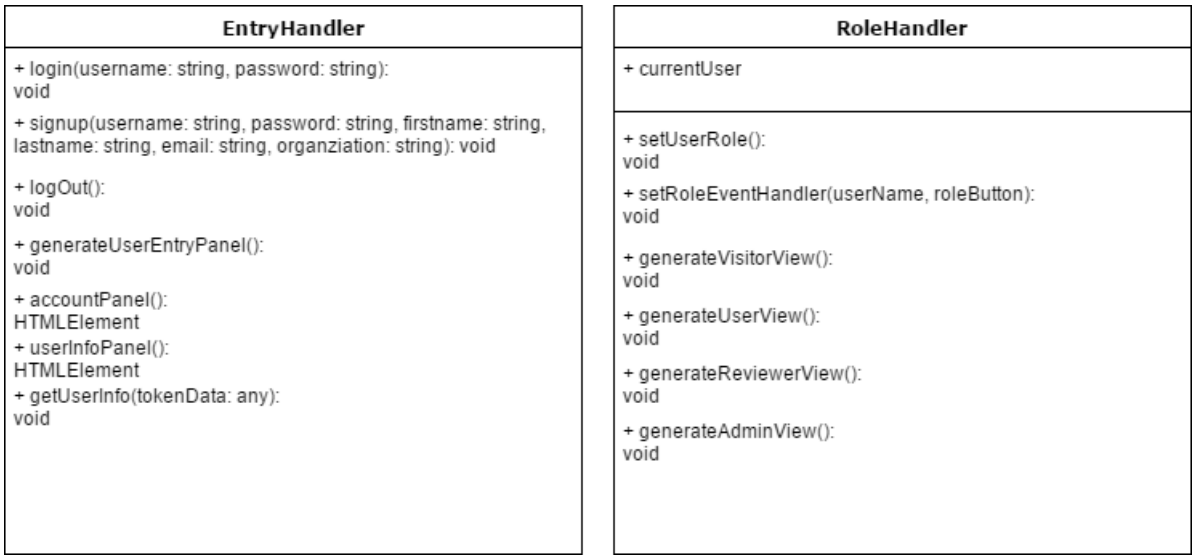


Figure 6.3: Class diagram of the EntryHandler and RoleHandler class.

6.3. Back-end Architecture

The back-end consists of several components: the API, the competitions module, the database, the configuration settings, data folders, and data readers. See below an overview of the back-end architecture:

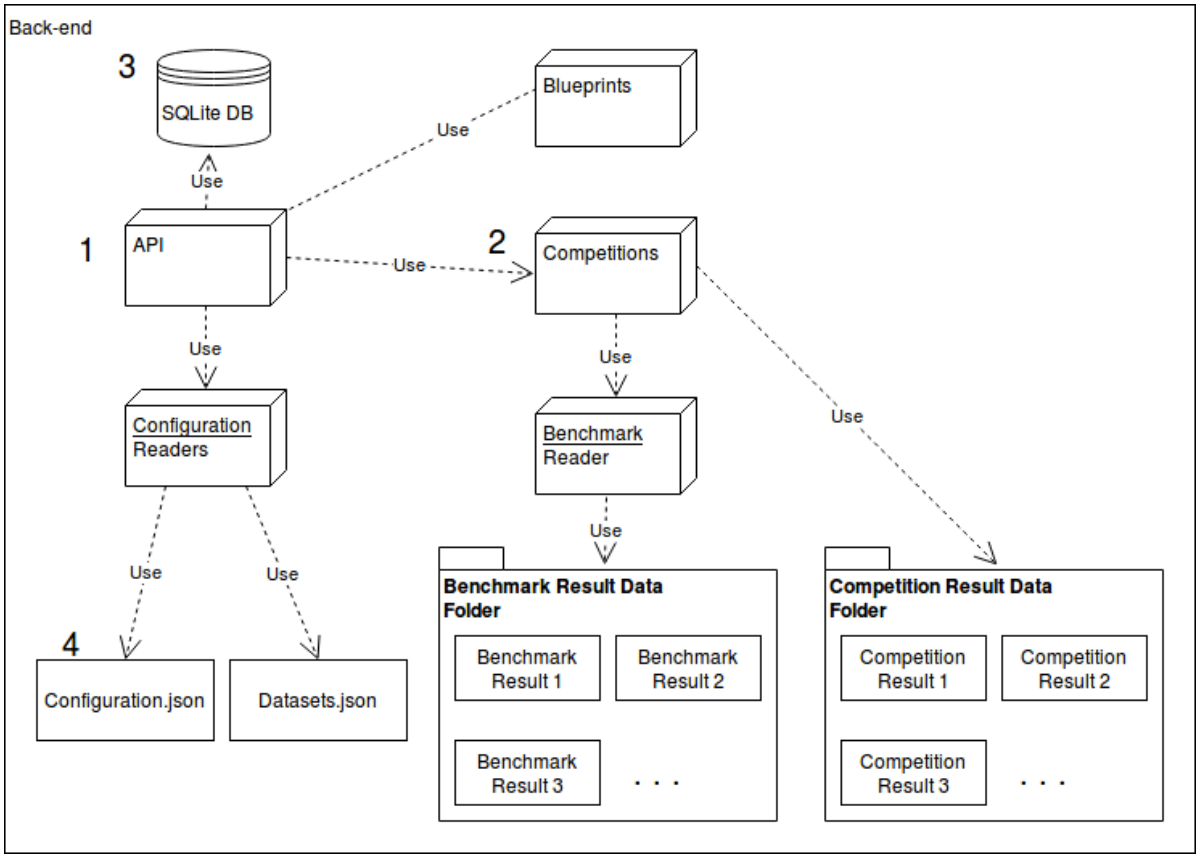


Figure 6.4: Overview of the back-end architecture. Each component numbered x is explained in section 6.5.x.

The exact format of the JSON data and its readers is not relevant and as such these components will not be addressed. The API, competitions module, database schemas and configuration settings for the website each have their section in this chapter. The API is explained in Section 6.3.1. The competitions module is explained in Section 6.3.2. The database schemas are explained in Section 6.3.3. Finally, the website configuration settings are explained in Section 6.3.4

6.3.1. API Overview

In this section we provide an overview of the API. To show which calls are available to which users of the website, we use a simple color coding:

Visitor Role: ○
Submitter Role: ●
Reviewer Role: ●
Admin Role: ●

These roles are the roles as discussed in Chapter 2. To quickly reiterate: the visitor role means available to anyone, with no authentication required, the submitter role requires solely authentication, and the reviewer and admin roles require specific granted permissions.
The API calls are organised by the functionality they deal with, pertaining to one of: benchmarks, competitions, participations, users, project information.

Benchmark

	Method	*URL*	*User*	*Request Body*	*Body Type*
1	POST	/api/benchmark/upload	● ● ●	Benchmark file	File
2	POST	/api/benchmark/remove	● ● ●	Benchmark Id	JSON
3	GET	/api/benchmark/data/<id>	● ● ●	-	-
4	POST	/api/benchmark/list	● ● ●	-	-
5	POST	/api/benchmark/list_all	● ●	-	-

Table 6.1: API calls pertaining to benchmark functionality.

Benchmarks can be uploaded and removed by users. Specific files can be requested through /api/benchmark/data/<id> by submitters if the file belongs to them, or a user can list all their benchmarks through /api/benchmark/list. Admins and reviewers can request a list of all benchmarks.

Competition

	Method	*URL*	*User*	*Request Body*	*Body Type*
1	POST	/api/competition/list	○ ● ● ●	-	-
2	POST	/api/competition/add	●	Competition Info	JSON
3	GET	/api/competition/open	●	Competition Id	JSON
4	POST	/api/competition/close	●	Competition Id	JSON
5	POST	/api/competition/publish	●	Competition Id	JSON
6	POST	/api/competition/unpublish	●	Competition Id	JSON
7	POST	/api/competition/run	●	Competition Id	JSON

Table 6.2: API calls pertaining to competition functionality.

Anyone is allowed to request a list of all competitions. The competition life-cycle, however, is completely controlled by the admins. /add, /open, /close, /publish and /unpublish all change a competitions state accordingly. /api/competition/run runs a competition, if it is closed, with as participants the accepted participations.

Participation

	Method	*URL*	*User*	*Request Body*	*Body Type*
1	POST	/api/participation/list	🟢🔴🔵	-	-
2	POST	/api/participation/list_all	🔵🔴	-	-
3	POST	/api/participation/add	🟢🔵🔴	Benchmark Id, Comp Id	JSON
4	POST	/api/participation/accept	🔵🔴	Participation Id	JSON
5	POST	/api/participation/reject	🔵🔴	Participation Id	JSON
6	POST	/api/participation/set_default_status	🔵	Participation Id	JSON
7	POST	/api/participation/withdraw	🟢🔴🔵	Participation Id	JSON
8	POST	/api/participation/unwithdraw	🟢🔴🔵	Participation Id	JSON
9	POST	/api/participation/assign_reviewer	🔵	Reviewer Id, Participation Id	JSON

Table 6.3: API calls pertaining to the participation functionality.

Participations are available to all authenticated users. A normal user can add a participation (using a benchmark and an open competition), request a list of their participations, and withdraw/unwithdraw their own participations. Reviewers can reject or accept participations. Admins can assign reviewers to participations and are also free to accept, reject or change the status back to default for any participation.

Users

	Method	URL	User	Request Body	Body Type
1	POST	/api/user/add	🟡🟢🔴🔵	User Info	JSON
2	POST	/api/user/remove	🔵	User Id	JSON
3	POST	/api/user/token	🟢🔴🔵	-	-
4	POST	/api/user/confirm/<token>	🟡🟢🔴🔵	-	-
5	POST	/api/user/get	🟢🔴🔵	-	-
6	POST	/api/user/list	🔵	-	-
7	POST	/api/user/promote/admin	🔵	User Id	JSON
8	POST	/api/user/demote/admin	🔵	User Id	JSON
9	POST	/api/user/promote/reviewer	🔵	User Id	JSON
10	POST	/api/user/demote/reviewer	🔵	User Id	JSON

Table 6.4: API calls pertaining to users.

Naturally adding a new user (registration) and confirming an account through a token link are available to unauthenticated users. Authenticated users can request tokens to substitute their login information with and also request their account information. Any user management - removing, viewing all users, and promoting/demoting accounts - is reserved for admins.

Project info:

	Method	*URL*	*User*	*Request Body*	*Body Type*
1	POST	/api/project/updates/list	●●●●	-	-
2	POST	/api/project/datasets/list	●●●●	-	-
3	POST	/api/project/docs/list	●●●●	-	-
4	POST	/api/project/repos/list	●●●●	-	-

Table 6.5: API calls pertaining to project information.

All project info is publicly available and can be requested by anyone.

6.3.2. Competition Module

The competitions module is where all the computation and organization of the competitions happens. See the following figure for an overview of the architecture of the Competitions module:

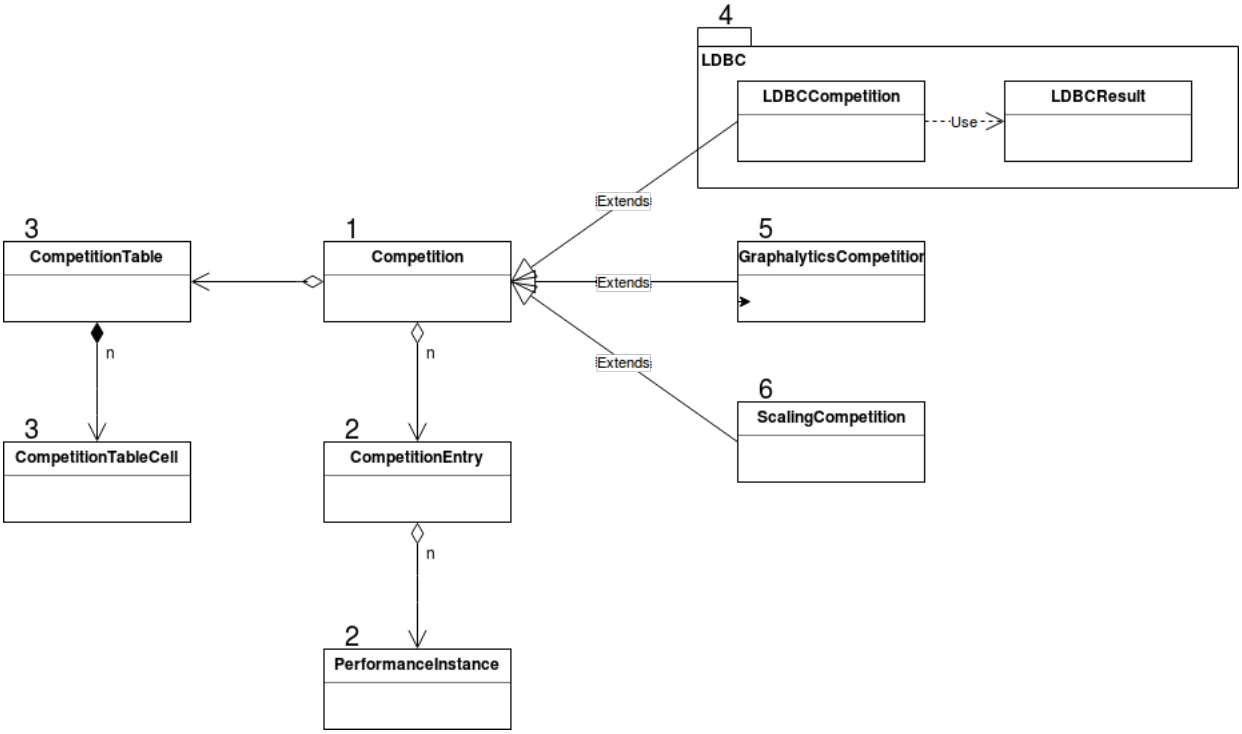


Figure 6.5: Overview of the back-end architecture. Each number x corresponds to the section 6.3.2.x where the class or component is discussed.

In this section we will go over each component of the competitions module. Starting with the central component: the Competition class. Then we will explain the objects the Competition class uses: the CompetitionEntry combined with the PerformanceInstance class, and the CompetitionTable class combined with its CompetitionTableCell class.

Competition Class

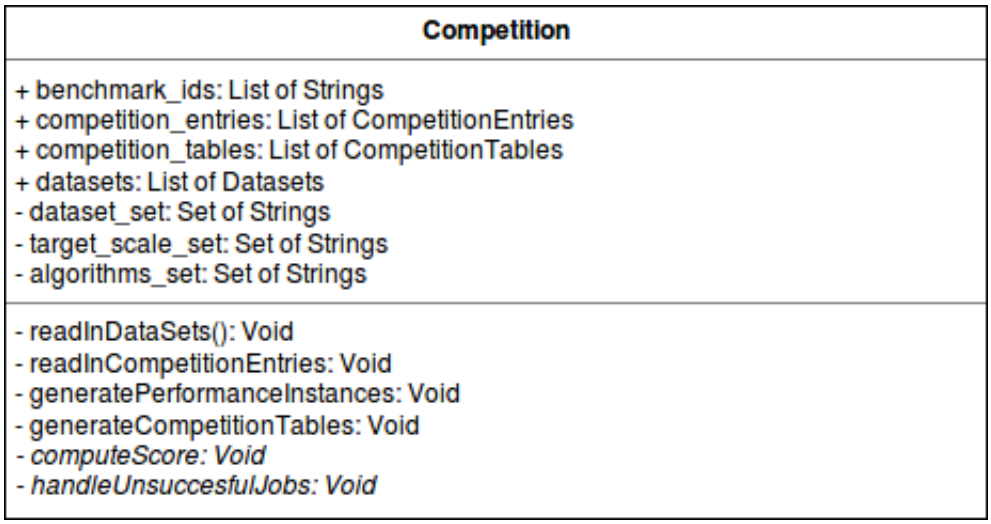


Figure 6.6: Class diagram of the Competition class.

The central class in the module is 'Competition'. The competition class carries the following responsibility:

- According to a list of competition ids of database competition objects, retrieve the corresponding benchmark result files from the upload folder, and read them in using a benchmark reader object. For each result file the reader reads, a CompetitionEntry object is created and stored in a list. The list of CompetitionEntries represents all the participants in the competition.
- Use a reader object to read in the dataset metadata (e.g. the amount of edges and vertices in the 'dota-league' dataset).
- Instruct each CompetitionEntry to generate their performance instances.
- Generate the CompetitionTables based on the CompetitionEntries it has.
- (Abstract) Compute the score. Actual types of competitions (LDBC, Graphalytics, Scaling) implement this functionality according to their format.

CompetitionEntry Class

The CompetitionEntry class represents a participant in a competition, it contains all the information of its corresponding benchmark result file. A CompetitionEntry carries the responsibility of generating its performance instances.

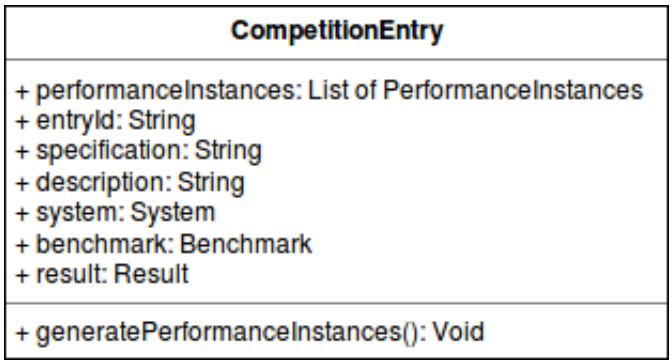


Figure 6.7: Class diagram of the CompetitionEntry class.

An example of a PerformanceInstance looks as follows:

Entry Id	*Algorithm*	*Dataset*	*EVPS*	*PPP*	*Load Time*	*Makespan*	*Processing Time*
b238415	BFS	dota-league	23502	0.00023	321.5s	123s	318s

Table 6.6: An example instance of a PerformanceInstance.

A PerformanceInstance represents an atomic instance of performance. It can be read as: System b238415 when running Breath-first-search on the dota-league dataset achieved an EVPS value of 23502, a PPP value of 0.00023, etc.

A system performs multiple runs for a particular combination of algorithm and dataset. For the load time, makespan and processing time, the PerformanceInstance stores the median of all the runs. The EVPS and PPP are calculated according to the median processing time.

CompetitionTable Class

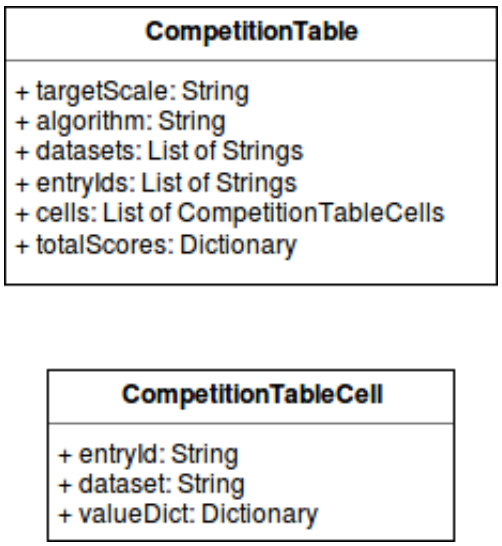


Figure 6.8: Class diagram of the CompetitionTable and CompetitionTableCell classes.

The CompetitionTable class represents a combination of an algorithm and a set of datasets in which systems compete. The class consists of CompetitionTableCells which have store a dictionary of the performance value for each performance metric, and its associated score. An example of a CompetitionTable instance, where the cells have been filled with the EVPS values from the dictionary:

	EVPS			
	dota-league	*graph500-22*	*datagen-7_9-fb*	*datagen-8_0-fb*
b238415	23472	12328	17289	12773
b581732	98142	82921	83492	-
b982521	12398	12399	-	-

Table 6.7: An example instance of a CompetitionTable.

The missing values in the table are the result of the systems failing those jobs in during the benchmark run. Each job performs 5 runs. Failure means that more than 2 out of 5 runs timed out. This value is stored in the settings file used by the system and can be easily modified to anything else. Our client requested it be set at 2 for now. CompetitionTables are used in competitions which score systems based on the performance of others systems for the same algorithm and/or dataset, such as the graphalytics and scaling competitions.

LDBCCompetition Class

To illustrate the results, and the differences between them, produced by the different competition formats, we will show the results of each competition on the same four systems: openg@das, graphx@das and two anonymous systems whose data is at this moment not to be revealed to the public.

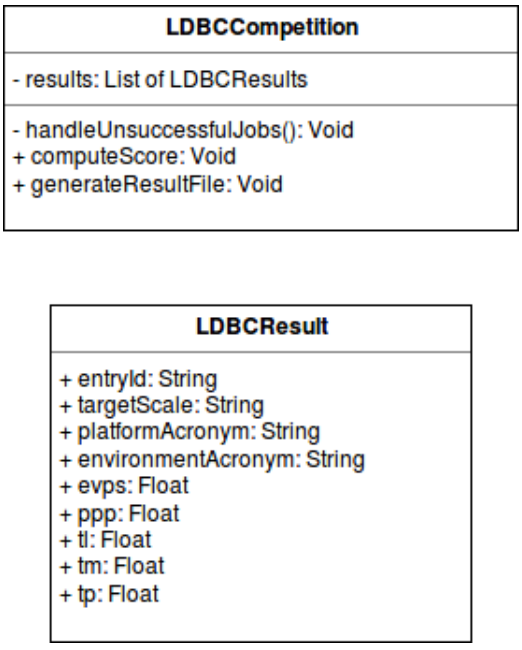


Figure 6.9: Class diagram of the LDBCCompetition and LDBCResult classes.

The LDBCCompetition class extends the Competition class and implements its method of computing the score according to its format. The LDBC competition assigns a single value of merit to each

participant in the competition. The systems are ranked according to their achieved PPP value. It does this by looping over all a system's performance instances and taking the mean of all the performance values a system achieved, or in the case of the PPP value, the harmonic mean.

Below is the result of the LDBC competition run on the four aforementioned systems:

Rank	Platform	Loading-time (TL)	Makespan (TM)	Processing-time (TP)	Performance (EVPS)	Cost Performance (PPP)
1	anonymous1	0.19000s	261.82s	153.82s	129 M unit / s	0.00000774 $ / unit
2	anonymous2	89.24833s	47.08s	21.05s	65.6 M unit / s	0.0000152 $ / unit
3	openg @ das	55.32223s	82.00s	75.83s	38.7 M unit / s	0.0000258 $ / unit
4	graphx @ das	5.20563s	382.89s	336.50s	1.52 M unit / s	0.000658 $ / unit

Figure 6.10: The results of LDBC competition for four systems.

GraphalyticsCompetition Class

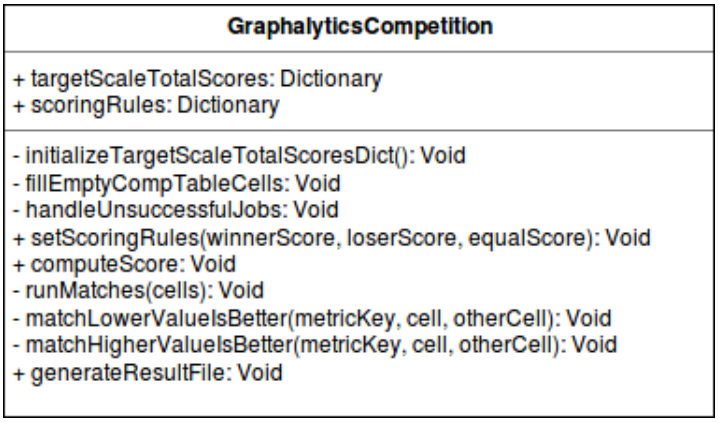


Figure 6.11: Class diagram of the GraphalyticsCompetition class.

The Graphalytics competition scores systems according to a pair-wise comparison to other systems. For each CompetitionTable it loops through each column, and 'runs' matches between the systems in such a way that each system is compared exactly once to each other system in each column. The 'winner' in a match is awarded 1 point, and the loser 0 point. In the event of a tie the score is 0.5 for both systems. In the case where a system has failed a job, it will automatically lose to every system for which the job succeeded, and automatically tie with every system for which the job also failed. The algorithm for computing the score can be seen in algorithm 1.

Each cell object keeps its own dictionary where it stores its performance values for each metric and its score. The algorithm modifies the cells' dictionaries in place, and as such, does not produce a separate output.

Algorithm 1: Compute the scores for each system according to the Graphalytics competition format.

```
ComputeScore (C)
    inputs : A list of competition tables C = c_1 ∧ ··· ∧ c_n;
    foreach competition table t do
        foreach dataset d in datasets(t) do
            cells = list(each cell in cells(t) where dataset(cell) equals d);
            i = 0;
            j = i+1;
            for cell c_i do
                for cell c_j do
                    if value(c_i < value(c_j) then
                        raise score c_i by 1;
                    else
                        if value(c_i > value(c_j) then
                            raise score c_j by 1;
                        else
                            raise score c_i by 0.5;
                            raise score c_j by 0.5;
                    j = j + 1;
                i = i + 1;
    return;
```

Below is the result of the Graphalytics competition run on the four systems:

BFS

System Name	datagen-7_7-zf	datagen-7_8-zf	datagen-7_9-fb	graph500-22	dota-league	Total Score (evps)
anonymous1	78316482 +3	67543592 +3	576543774 +3	449678324 +3	863245474 +3	15
anonymous2	70944097 +2	64878400 +2	227900811 +2	253050920 +2	363796307 +2	10
openg@das	62802971 +1	64732442 +1	130131704 +1	101143452 +1	132289566 +1	5
graphx@das	798772 +0	757488 +0	4010046 +0	3483141 +0	9734610 +0	0

Figure 6.12: The results of the Graphalytics competition for four systems.

ScalingCompetition Class

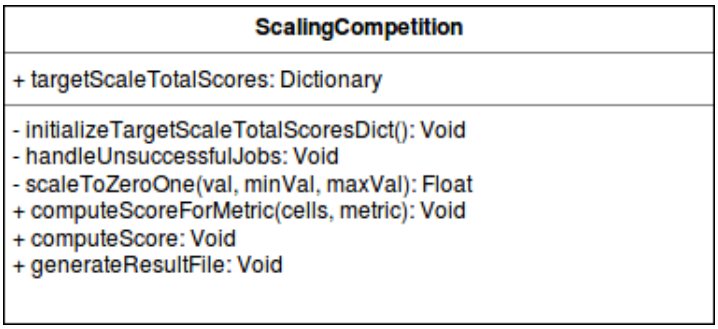


Figure 6.13: Class diagram of the GraphalyticsCompetition class.

The Scaling competition works very similarly to the Graphalytics competition: systems also receive a score per column in a CompetitionTable. The difference lies in how that score is computed. The Scaling competition format loops over each CompetitionTable over each column. For each cell in a column it scales the performance value to a value in the interval [0, 1]. The scaling is based on the lowest and highest performance values achieved in that column. As such, the system with the lowest performance value will receive a score of 0, and the system with the highest performance value a score of 1.

Below is the result of the Scaling competition run on the four systems:

BFS

System Name	datagen-7_7-zf	datagen-7_8-zf	datagen-7_9-fb	graph500-22	dota-league	Total Score (evps)
anonymous1	78316482 **+1.000**	67543592 **+1.000**	576543774 **+1.000**	449678324 **+1.000**	863245474 **+1.000**	5
anonymous2	70944097 **+0.905**	64878400 **+0.960**	227900811 **+0.391**	253050920 **+0.559**	363796307 **+0.415**	3.23
openg@das	62802971 **+0.800**	64732442 **+0.958**	130131704 **+0.220**	101143452 **+0.219**	132289566 **+0.144**	2.34
graphx@das	798772 **+0.000**	757488 **+0.000**	4010046 **+0.000**	3483141 **+0.000**	9734610 **+0.000**	0

Figure 6.14: The results of Scaling competition for four systems.

6.3.3. Database Schemas

In this section we provide an overview and explanation of the database schemas the website uses. The main functionality of the website concerns four schemas: users, benchmarks, competition and participations.

User		
Column	*Type*	*Attributes*
id	integer	primary key
username	string	indexed
password	binary	non-nullable
admin	boolean	-
reviewer	boolean	-
forename	string	-
surname	string	-
email	string	-
organization	string	-
status	string	-
registered_date	date	-
confirmed	boolean	-
deleted	boolean	-

Table 6.8: Database user schema.

The users table naturally stores the username, password hash and email of the user. It also stores some additional info like the user's real name and affiliated organization. The status can be active or inactive. The user needs to be confirmed before he/she can perform any meaningful actions like uploading a benchmark result file. The role of a user is denoted by the admin and reviewer flags, which grant the permissions to perform certain actions as described in Chapter 2. Finally there is a flag for if a user has deleted their account. When a user deletes their account, their information is kept in the database. This is because their information might be linked to published benchmark results which cannot be removed.

Benchmark		
Column	*Type*	*Attributes*
id	integer	primary key
benchmark_id	integer	-
acronym	string	-
target_scale	string	-
specification	string	-
owner_id	integer	-
upload_time	datetime	-
status	string	-

Table 6.9: Database benchmark schema.

The benchmark result file a user uploads has an associated benchmark object in the database. The benchmark id is the name of the folder which contains the corresponding result file. The owner id links it to the user who uploaded it in the user schema. Some additional information about the

benchmark result like its target scale and specification are also stored. For a description of the status and the life-cycle of benchmark objects in the database, see Section 5.4.1.

Competition		
Column	*Type*	*Attributes*
id	integer	primary key
name	string	-
start_time	string	-
end_time	string	-
specification	string	-
status	string	-
type	string	-

Table 6.10: Database competition schema.

Competitions are kept track of using the competition schema. This schema is the most straightforward, as all the competition information is stored as strings. One thing to note is the start and end time are not date or datetimes. This is because this input needs to come from an admin through the front-end. The implementation of which required a datepicker input which proved to be hard to find an implementation of that was compatible with Typescript. Since the start and end time are only shown to users and no actual computation is done with them, we have decided to leave them as strings. This issue is included in our compiled list of issues. The status and life-cycle of competitions in the database are described in section 5.4.3.

Participation		
Column	*Type*	*Attributes*
id	integer	primary key
user_id	integer	-
benchmark_id	integer	-
competition_id	integer	-
ranked	boolean	-
status	string	-
reviewer_id	integer	-

Table 6.11: Database participation schema.

Participations record which benchmarks compete in which competitions. The benchmark id and competition id link to the corresponding objects in their respective schema. The user id links it to the owner in the user schema, whilst the reviewer id links it to the assigned reviewer in the user schema. The status and life-cycle of participation objects are described in Section 5.4.2.

6.3.4. Website Configuration

The website uses a configuration file. In this section we provide an overview of the settings which can be configured.

- secret_key: The secret key the website uses to create tokens.
- port: The port the server should run the website on.
- sql_db: The path to the directory containing the SQLite database.
- text_db: The folder containing certain textual data to be displayed on the website.
- mail_server: The mail server used to send account confirmation emails.
- mail_port: The mail port to use.
- mail_use_tls: True or False.
- mail_use_ssl: True or False.
- mail_username: Username for the email account of the website.
- mail_password: Password for the email account of the website.
- num_successful_runs_required: The amount of runs required for a job to succeed.
- source_dir: The path to the directory which contains the uploaded benchmark result files.
- result_dir: The path to the directory which contains the competition results.
- dataset_metadata: The path to a json file containing the metadata for each dataset e.g. the amount of vertices.
- benchmark_result_filename: The filename to use for an uploaded benchmark result file.
- ldbc_result_filename: The filename to use for the result file of an LDBC competition.
- graphalytics_result_filename: The filename to use for the result file of a Graphalytics competition.
- scaling_result_filename: The filename to use for the result file of a Scaling competition.

6.4. Security

In this section we will briefly touch on the security of our implementation. By discussing possible attack vectors.

6.4.1. Cross-site Scripting

Because the validation of benchmark result files is not implemented, the website is currently vulnerable to cross site scripting. Although a user is only able to upload JSON files, in theory it might be possible for a user to include a script as the value of a JSON attribute. Although it is hard to predict what exactly would be possible in such a scenario. The validation of the JSON would close this attack vector however.

6.4.2. SQL Injection

Any "special" characters (such as semicolons or apostrophes) in data are be automatically quoted by the SQLEngine object used by SQLAlchemy. This makes SQL-injection attacks basically impossible.

6.4.3. Rainbow Table Attack

Passwords are salted using bcrypt, so using a rainbow table will not give an attacker an advantage.

6.4.4. Brute-force Search

We have a minimum requirements of 8 characters for passwords which makes brute-force searching impractical. Moreover, the bcrypt hashing algorithm is deliberately slow, and uses an iteration count which determines how often the hashing process should be repeated. This iteration count can be adjusted in the website settings by the system administrator to adjust difficulty.

6.4.5. Data Breach

In the case of a data breach the name, email and organization of users would be exposed. Although this is not desirable, this is not particularly harmful. Only hashed passwords are stored, which means users' passwords will remain safe. The only sensitive information that exists on the server are the benchmark result files.

7

Quality Assurance

During our research phase of this project, we have made a plan to employ several practices so that the code quality is kept as high as possible. In this chapter we evaluate how the plan was carried out during the course of this project. First, Section 7.1 provides our initial plan to ensure high code quality. Then, in Section 7.2, we provide information on how Automated Static Analysis Tools (ASATs) were used in this project and how we have documented our code. In Section 7.3, we go into detail on how we tested our system and how continuous integration was used. Lastly, in Section 7.4, we discuss the score that we have received from Software Improvement Group (SIG) and how we have processed the feedback.

7.1. Initial Plan

To ensure that the quality of the code is kept as high as possible, during our research phase we designed the following plan:

- TSLint [22] and PyLint [12] will be used for statically checking the code quality.
- The feedback of SIG will be used to improve the code.

And for the testing and continuous integration we came of with the following plan:

- Mocha mochajs.org [14] and Chai [4] will be used to used to write tests for the front-end.
- *unittest* will be used to used to write tests for the back-end.
- Travis-CI [25] will be used for continuous integration.

7.2. Code Quality

In this section we analyze the usage of automated static analysis tools, how we have documented the code, how we have tested our system and how we have used continuous integration in our project. ASATs are tools that are used to statically check your code for defects without running the code. Code documentation is where the developer writes a description for the code they have written. This documentation can help other developers understand the functionality of the code. To confirm the expected behavior of the system, test is written for the system. Continuous integration builds source code in a "clean" environment. This can help developers to avoid running into issues where a functionality only works on their machine and not for others.

7.2.1. Usage of ASATs

As mentioned in Section 7.1, we have decided to use TSLint and PyLint to statically check our code. We have not deviated from this plan, but we have used an extra tool, pep8, for code that is written for the back-end. *pep8 is a tool to check your Python code against some of the style conventions in PEP 8.* [20]. A PEP 8 checker is included in PyCharm, so there was no need for us to manually install it ourselves.

Though we have resolved most warnings, there are still warnings that remain in both the front-end and the back-end. We have decided to not resolve the remaining warnings, as they do not cause functional defects. Therefore these warnings had a lower priority. A possible reason why we got these warnings could be that we used the default configuration of the tools. We think that we would have gotten a smaller list of warnings (or even no warnings at all) if we have used a custom configuration of the tool, because then the tools would have only shown relevant warnings; in other words, the remaining warnings could largely be ascribed to pedantic checking.

7.2.2. Code Documentation

Not all classes in the front-end are well documented. This is due to the change in the plan that is explained in Section 8.4.1. Getting the required features working had a higher priority, which reduced the time for writing and checking more documentation. For the code that we have written in the back-end, not all classes are properly documented. For new API calls that we have added, we did not provide documentation as we think that the function names give a good description what they are (each) supposed to do. For the new classes that have created, especially the one that are used to represent JSON objects are not documented. We did not provide documentation as these classes do not have any functions; they are used to store data that is required for computing the competition result.

7.3. Testing and Continuous Integration

7.3.1. Testing Front-end Code

The original plan was to use Mocha and Chai to write tests for the front-end. We did use both libraries to write tests for the front-end, but we have also used two other libraries: Sinon.JS [5] and jsdom-global [6]. Sinon.JS was used to create mock objects so that we can test functions where an AJAX call is made with jQuery. An AJAX call requires an Internet connection and a browser for it to work, it is therefore useful to use Sinon.JS to simulate an AJAX call.

We used jsdom-global to simulate DOM in the Node.js environment. With this approach we were able to generate HTML elements in our tests without having to run a web browser. This was useful for testing the different functions within a Page class.

We wrote tests for the front-end until we had to adjust our planning. After adjusting our planning (see Section 8.4.1), we have decided that writing tests for the front-end does not have a high priority. We have made this decision because of the following reason: the front-end code is used to render the different pages in the web browser. We can see whether a page is rendered correctly in the web browser by checking whether all element are rendered as expected. The front-end therefore serves as a GUI and writing test for the front-end is like writing test for a GUI. Writing these tests will cost us time that can be used to develop new features. Therefore we have decided to not write any more tests for the front-end.

7.3.2. Testing Back-end Code

For the back-end we originally planned to used *unittest* to write tests for the back-end. We have not deviate from this plan and we have written tests for the back-end using *unittest*. Though we did not test everything thoroughly, we have tested the most important components of the back-end, which are how the competition results are computed.

7.3.3. Test Coverage

The line coverage of the front-end code was at 100% in the first few weeks of our development phase. It then dropped as we implemented new features in the front-end. These new features requires an Internet connection and a web browser for them to work. We were able to create mock objects in our tests to simulate the functions being called, but this does not cover the code within the function itself. Also the tests are run on the JavaScript files that has been transpiled from the TypeScript files, therefore there are lines that were are not able to cover. The line coverage of the front-end at the end of this project is at 44.57%. This is a result of the decision that is mentioned in Section 7.3.1. Test coverage of the front-end can be seen in Fig. C.1, Fig. C.2 and Fig. C.3.

As for the back-end we have tried multiple ways to get the test coverage working on Travis-CI, but we had no luck. The problem was that Travis-CI was not able to find our tests. We therefore cannot provide any information regarding the test coverage of the back-end.

7.3.4. Usage of Continuous Integration

As mentioned in Section 7.1, we have decided to use Travis-CI for continuous integration. We followed the plan and used Travis-CI to build our code in a "clean" environment. We initially had two separate repositories, one for the front-end and one for the back-end. Travis-CI was configured for each of the repository. We eventually decided to merge the front-end and back-end into one single repository. Because of the problem mentioned in Section 7.3.3, we configured Travis to only run on the front-end. We used Travis to check the build status of a pull request. No pull request were approved if the build did not pass.

7.4. SIG Code Evaluation

All teams are required to upload their code twice to SIG. The code is checked by SIG and an overall score for maintainability is given based on different metrics. The lowest score that can be given is one star and the highest five stars. The first upload is to provide feedback to the team how they can improve the maintainability of their system. The second upload is for the team to assess how well they have improved the maintainability of their system based on the feedback they have received.

7.4.1. Score of First Upload

We have received a score of three and a half stars out of five for our first upload to SIG. This means that the maintainability of our system is above average. The original feedback (in Dutch) is available in Appendix B. We have not received the highest score because we scored low on code duplication and unit size.

For code duplication, SIG looks at the percentage of duplicated code in the system. The lower the percentage, the lower the score. Duplicated code were found in e.g. *GraphalyticsCore.py* and *GraphalyticsDriver.py*.

For unit size, SIG looks at the percentage of code that are too long (e.g. long methods/functions). Same as for code duplication, the score is better if the percentage is lower. Large unit size were found in e.g. *computeScore()* in *GraphalyticsCompetition.py*, the code for creating *"curr_alg_p_instances"* and the code for creating *"dataset_dict"*.

We have also received feedback that our amount of tests that we have written looks promising and SIG hopes that the amount will by the time we have implemented new functionalities.

7.4.2. Processing SIG Feedback

We initially want to improve our code based on the feedback that we have received from SIG, but did not manage to do so. This is due to our new planning (see Section 8.4.1). After discussing with our client, we have agreed that developing and getting the remaining features have a higher priority. We have therefore not spend a lot of time to improve our code based on the feedback.

7.4.3. Score of Second Upload

For our second upload, our score did not change from the first upload. It stayed at three and a half stars out of five. The feedback of the second upload can also be found in Appendix B. SIG noted that we have resolved the code duplication in the files that were mentioned in Section 7.4.1. But due to the new classes that have been added later on in the project, the percentage of code duplication stayed almost the same. Files that contained high percentage of code duplication were the following: *AdminSubmissionTableGenerator.ts*, *GraphalyticsCompetition.py* and *ScalingCompetition.py*.

There were no comments on Unit Size, but we assume that the feedback in the first upload still holds. As mentioned in Section 7.4.2, we did not spend too much time to improved our code based on the feedback.

SIG have noticed that the amount of test code has increased. This is positive as we have added more classes. Though the amount of test code has increased, we are still falling short with the amount of test code, but this is due to the decision that is mentioned in Section 7.3.1.

8

Project reflection

In this chapter, we discuss on the following: (1) our workflow, (2) how we communicated internally as a team and externally with our client, (3) the problems that we have encountered throughout the course of this project and (4) how we have dealt with the problems.

8.1. Workflow Evaluation

As mentioned in Section 4.1, we chose to use adapted version of the Scrum to manage our development processes. We have not deviated from this plan and all development processes was managed using the adapted development methodology.

Every Monday, we had an internal meeting to create the new sprint plan. It was until sprint seven that we have written more detailed sprint plans: a bigger task is broken down to smaller tasks. Each task in the previous sprint plans had implications that it contained several smaller tasks. These were not explicitly written down in the sprint plan. We changed to writing detailed sprint plans due to the following reasons:

1. We have received feedback from our client that it was a bit difficult to estimate how much time each team each member spending on their tasks or how large a task is.

2. As mentioned in Section 4.4.3, a team member was going to work remotely from China. Because of the different time zones, good communication is needed in order to make sure that a team member knows what their tasks are.

Every Friday, we had a meeting with our client to discuss how the sprint went: we update our client on what we have been working on and whether we have encountered any issues while working on our tasks. During the meeting, the feedback that is given by our client is noted down and are used when we are planning the next sprint.

One thing that we can take away from this, is that we should have written detailed sprint plans from the beginning. This have benefits for both the team and the client.

For the team:

- Each team member has a better overview of all the task that has been assigned to them.
- There is no need for a team member to figure out whether a task contains several smaller tasks.
- Each team member can manage their time better by having a better overview their tasks.

For the client:

- Better overview of all the tasks that has been assigned to each team member.
- Can have a better estimate how much time each team member will be spending in the sprint.

8.2. Communication

8.2.1. Communication within the Team

Throughout the project the team communicated mainly through Telegram (an instant messaging application) and a few times through Skype. We used Telegram to update each other on the progress of the tasks: on which task is a team member working and whether problem has been encountered. Skype was used for updates that were not easily communicated through texting.

8.2.2. Communication with Client

We communicated with our client mainly through weekly face-to-face meetings and through Slack. The weekly meetings took place at the end of the week, as mentioned in Section 8.1. We used Slack to communicate with our client when our client is not able to meet with us physically.

8.2.3. Communication with Academic Coach

Communication with our academic coach went mainly via face-to-face meeting, Slack and Skype. We used face-to-face meetings to update our on the progress of the project. We used Slack and Skype when our academic coach were not able to meet with us physically.

8.3. Problems Encountered

During the course of this project we have encountered the following problems:

- Falling behind schedule.
- Compatibility issues between libraries and TypeScript.
- International Communication.

8.3.1. Falling Behind Schedule

During our midterm review with our academic coach and our client, we have discussed that we were behind in schedule with development of our product. The reason why we have fallen behind schedule is due to our inexperience with web application development. During the first few weeks of development, we also need to spent time to research on how everything works e.g. figuring out how to test the code that is written for the front-end. This has caused us to lose some time on developing working features.

In Section 4.4.1 we mentioned that the most challenging part for us would be the back-end development, but it turned out that the most challenging part for us was the front-end development. This was also a reason that had caused us to fall behind in schedule.

8.3.2. Compatibility Issues between Libraries and TypeScript

While we were working on the front-end we were looking for libraries that can aid us with the implementation of certain features. An example would be the following: a library that can create table headers that sorts the rows of the table when the user clicks on a column.

This allows the table to be more interactive. We managed to find a library that does exactly what we want. The library is called tablesorter [2].

In order to get this library working with out TypeScript code, we first need to install it via Node Package Manager (npm)[16]. After installing the library, we have tried to create the same example table that was shown in the documentation. Unfortunately we weren't able compile the TypeScript code. We have tried to import the library as a JavaScript file, but this also did not work. The reason why this didn't work is because TypeScript works with types and the library does not have a type. This causes a conflict between the library and TypeScript.

8.3.3. International Communication

A team member worked partially remotely, due to being located in China between July 8th and August 17th. For the first six weeks of the research and development period all three of team members were working together in the office provided by the Graphalytics team. The main way of communication during this period was face-to-face. After the remote working period started, the communication within the team was done mainly through Telegram and Slack. This made the communication within the team indeed more difficult than during the first couple of weeks, but since it was already discussed at beginning of the project within the team, with the client and the academic coach, our team has made plans to tackle the problem. There were several difficulties with working remotely in another country. First, there is six hours of time difference between Netherlands and China. It is hard to find a common time. Second, the Internet connection was not ideal at the hotel. Third, due to the Internet censorship, Google Drive did not work in China without the VPN; and VPN approaches go near or straddle the border of legality under Chinese law. This made sharing of documents within the team more difficult.

8.4. Addressing the Problems

8.4.1. Falling Behind Schedule

During our discussion with our academic coach and our client, our coach had asked whether it is possible to get back on track by working harder and spending extra time on the development. Our coach had also suggested that we need to have a meeting with the client to discuss whether there are requirements where their priority can be lowered. This can help us focus more on the essential requirements of the product.

During our weekly meeting with our client, we went over the requirement (see Section 2.2) and had a discussion on creating a new requirement list with their updated priorities. The requirements where their priorities has been lowered (to being a could have feature) are the following:

Requirements for the role of a viewer:

- A viewer can sign up using CAPTCHA.
- A viewer cannot sign in more than x time in y amount of time.
- A viewer can update their profile information.

Using CAPTCHA was to verify that the user is human and not a bot. The priority of this feature has been lowered due to another feature that has been implemented: viewer can sign up using verification via email. The priority of the second feature in the list has been lowered due to a library, bcrypt, that we have used to process a login of a user. Verifying the passwords with bcrypt takes time as it first needs to be hashed and then salted. Due to the time that is needed to verify the password, it prevents a user from trying to login multiple times within a time period. After discussing with our client, we have come to the conclusion that the last feature in the list would be handy for the user

of the website, but it is not a critical requirement for the system. The priority of this feature has therefore been lowered.

Requirement for the role of a admin:

- An admin can restore the system programmatically and manually.

After discussing with the client, we have to come to the conclusion that it is nice to programmatically restore the system but given the remaining time we had for the project it has been moved to a lower priority, though we do have to provide a manual on how to manually restore the system.

As it had turned out that the front-end development was the most challenging part, instead of overestimating the amount of time needed for a feature in the back-end, we overestimated the amount of time that is needed for a feature in the front-end. We also spent extra time working on the front-end in order to get back on track with the schedule.

8.4.2. Compatibility Issues between Libraries and TypeScript

Fortunately we did not encounter compatibility issues that causes an essential feature of the system to not be implemented. Most of the compatibility issues had to do with a library that we want to use to improve the user experience. After discussing with the client, we have decided that these issues were minor issues and do not need an urgent fix.

We did not expect encounter these compatibility issues when we had decided to use TypeScript to develop the front-end. We initially assumed that TypeScript should allow the import of JavaScript files as it is a superset of JavaScript, but this has shown that our assumption was wrong.

What we have learned from using TypeScript is that the type checking feature can you help you find programming errors before you compile you code. But it seems that TypeScript might not be a good programming language to use for front-end development due to the compatibility issues we have encountered.

8.4.3. International Communication

We have discussed in advance to solve the communications issues. First, for the time difference, we have agreed to communicate through Telegram and have fixed time period when all of team members are available. Since China is six hours ahead of the Netherlands, 10:00 A.M. in Netherlands is 16:00 P.M. in China. Then all of the team members would be present on Telegram and Slack. Second, when Internet connection was not good enough, meeting notes were written by a team members and are communicated through Telegram afterwards. Messaging is not the only way that we communicated with each other, Telegram voice call was another quicker way to update each other. All meeting were face-to-face meetings up until week six. From week seven, meetings two team members were physically present and the other was present through Skype.

8.5. Meeting the Requirements

In the last week of development, we have sat down with our client and discussed whether we have our product has met the requirements. Some could have features were not met, but these did not have a high priority.

There were two should must have features that were partially done and three should have feature that did not manage to finish:

Must have features:

- viewer has access to the links to the documentation and repositories relating to the project.
- viewer can see the project organization and people involved.

These features are partially finished as the data are not stored on the database. The plan was to insert all the data into the database, so that they are not stored in a JSON file anymore. This process was time consuming and given the fact that we were behind schedule, we decided to keep the data stored in the JSON files. We will have to leave this as future work.

Should have features:

- reviewer can explain a rejection in a comment.
- the system can update submissions' report format for backwards compatibility in competitions.
- the system can verify the syntax and missing fields of a benchmark submission.

After discussing with the client, we have come to the conclusion that the first feature in the list was not an essential feature that will cause the system to not work. The other two features however, were features that our client wanted to have in our product. It was unfortunate that we did not manage to implement these two features, but there were other features with higher priority and we were running behind schedule.

8.6. Learning from this Project

There are several things that we have learned from this project:

- As mentioned in Section 5.2, we went for a single-page design. But while developing the front-end we have noticed that this design had lead to the following issue: redirecting to a particular page on the website. The front-end has one main script that starts the application and instantiates all the necessary objects. This script will always start at the main page and it is difficult to start at another page by pasting the link into the browser. Though this issue was resolved, it had cost us some time to solve it.
- In the beginning we did not make a design for the user interface of the website and for the life-cycles of the database objects. This has caused us to spend time figuring out what would be a good design while we were developing the product.
- In the beginning we have spent quite some time researching how we can test for the front-end. We should have realized sooner that it should have been manually tested, as the front-end serves as the GUI of our system. This is a reason that had caused us to fall behind schedule.
- We have realized that TypeScript might not be a good programming language to use to develop the front-end of our system. This is due to the compatibility issues that we enountered.

If we would have to redo this project or do a similar project in the future, we would not go for the same design for the front-end: we would use multiple HTML files instead. We would first make a design user interface of the website and the life-cycles of the database object, so we do not have to figure out the design while developing the product. Also we would definitely not spend so much time figuring out how we can test the front-end. And we would choose to use JavaScript to develop the front-end.

9 Conclusion and Future Work

After finishing with our project, we have learned some important lessons during the process. In this Chapter, we will first give the conclusion for the whole project in Section 9.1. Then, in Section 9.2, we discuss ethical issues and the teamwork. Finally, in Section 9.3, we talk about the potential new features which could be added in the future.

9.1. Conclusion

In this bachelor project, we have developed a website, that our client can use to host competitions. The main contribution of the this website and the work done for the thesis can be concluded as following:

1. Research and design a fair scoring system for the Graphalytics competition.
2. Implement website to promote and host Graphalytics competition.

Our main research question was the following:

> *How can we create a sensible competition that is fair for all participants, and assigns each participant a score that has merit?*

Our answer to our research question is the following:

We wanted our competition format to use multiple categories and no penalization of weakly performing systems, two characteristics the Graphalytics competition format has. Moreover, the Graphalytics format scores systems amongst each other per combination of algorithm and dataset. Comparing two systems' performance on two differing algorithms or two differing datasets is not fair.

This is why we decided to base our Scaling competition format on the Graphalytics competition format. The main difference is how we score the systems based on their performance given a combination of algorithm and dataset.

And our answers to the subquestions are the following:

> *1. (conceptual) What metrics should be used to rank the participants?*

The main metrics used to evaluate participants are: EVPS, PPP, Loading Time, Makespan and Processing Time. Particularly EVPS and Processing Time provide more information on the performance of a graph analysis platform.

2. (conceptual) Which possible scoring formats exist and what are their advantages and disadvantages?

We have looked at several different real-world scoring formats some of which belong to other graph benchmark result ranking competitions like Graph500 and Top500, and others which belong to other kinds of competitions like SYNTCOMP and RoCKIn which concern robots. We have seen that very few competition formats seem to use a single value of merit to rank participants on. Instead, most of the competitions we have found rank their participants in multiple categories, foregoing the difficulty of appointing an overall winner. The usage of performance values directly as score and the lack of penalization for weak performance are also characteristics shared among the large majority of the competition formats we have looked at. The LDBC and Graphalytics competition formats, which our client provided us with for implementation in the Graphalytics website, also share these characteristics. The exception being LDBC which specifically uses a single value of merit.

3. (conceptual) Is it possible to summarize the score from different metrics in a representative single value of merit?

The LDBC competition format summarizes the total performance of a system into a single value of merit, namely the price per unit of computation. Although the LDBC competition also presents viewers with the other performance metrics EVPS, loading time, makespan and processing time as average values, the ranking of participants is based purely of the PPP value. In the end, the PPP value does provide the user with a general indication of how much a system would cost to use on average over a myriad of different algorithms and datasets. In this sense, the value has merit. For anyone looking to process very specific graphs or use very specific algorithms, the value has less merit. For such a person it would be user to look at the results of a competition format which looks at different categories, and try to find the category which most matches their situation.

4. (technical) What are the main stakeholders and requirements for the website?

Based on analysis that we have done in Chapter 2, the stakeholder are the following: System Integrators, System Developers, System Buyers, Hardware Vendors, Academic Researchers, Standardization Body and Project owners. The requirements of the website is listed in Table 2.2

5. (technical) How to design a system for a global competition of graph-processing platforms that supports the stakeholders and meets the project requirements?

The system that we have designed consist of two main components: a front-end and a back-end. The front-end servers as the GUI of our system, and the back-end handles all the API calls and database operations. For the front-end we went for a single-page design to avoid code duplication in our system. To satisfy the requirements where a role is involved, we have designed role-specific views in the front-end.

6. (technical) How to implement and test the system from point 5?

We implemented the front-end of our system using TypeScript, HTML, CSS and Bootstrap. We used Mocha, Chai and Sinon to automatically test the front-end. We have also done manual testing for the front-end. We implemented the back-end of our system using Python. We used *unittest* to automatically test the important components of the the back-end.

We have encountered several problems in the project, but fortunately these problems were resolved after discussing with our client and academic coach. We initially started this project with barely any experience of web development. With this project we have learned several lessons and gained more web development experience. These can be used for future projects that are similar to this one.

9.2. Discussion

In this section, we first discuss the ethical issues involved in the project, and then evaluate the teamwork during the course of the project.

9.2.1. Ethical Issues

The two most important issues that are relevant to the website that we have created are the following: privacy and security. Concerning privacy issues are the benchmark results that are uploaded by the user of the website. Not all user wants their benchmark results to be made public, as the results might contain confidential information of their system. Publishing these confidential results to the public might cause huge issues between our client and the owner of benchmark results. The website only uses the benchmark results to compute the results of the competition, but what our website is missing is to provide option that allows the user to choose whether a benchmark result is allowed to be shown to the public.

Concerning security issues is data breach. Again the most important data are the benchmark result files of the users. If a hacker is able to breach into our database, it is possible for them to leak the benchmark result files. This can cause also cause huge issues between our client and the owner of the benchmark results. We have tried to make back-end as secure as possible with the technologies that we have used, but no technology can provide protection against every attack.

9.2.2. Teamwork

All three of our team members are quite familiar with each other, and we have been working together for three years now. So communication within the team is quite good, and we can talk freely to each other about what could be done and what our thinking processes are. During the course of the project, our client has provided us an office. In the office we were able have a discussion by means of a face-to-face meeting.

We were able to update each other about our progress and difficulties easily through the help of Telegram or Slack when we were not at the office. We also distribute our work evenly, and also help each other out.

Due to the unique situation of our project taking place over the summer vacation, and as discussed in 4.4.3, two team members stayed in the office, and one worked remotely (after week six). The teamwork is maintained by communication through Telegram and Skype. Any updates and new tasks are discussed thoroughly with each other, either through text messages or voice calls. So from our experience, face-to-face meetings are efficient, but not necessary to maintain a good teamwork. Remote working is still viable when there is good communication within the team.

9.3. Future Work

Though we have met almost all the requirements that are listed in Section 2.2, we did not have enough time to fine-tune our system. The user is user-experience on our website is not the best. There are still some remaining issues left for the future implementation. Ther are listed in our GitHub repository. The are also some new functionalities and aspects which can be implemented or improved upon for any other developers who wish to continue with this project in the future.

We came up with a competition ourselves that uses a different scoring method. But more research can be done to see whether there are other scoring methods or even another different competition format that can be used on the Graphalytics website.

The benchmark results of Graphalytics competition give the users and developers better understanding and more insightful knowledge of the graph-processing platforms. However, it might also contain sensible information such as environment specification or non-default configuration options [9]. Therefore it would be a nice to provide the user the option to select whether the benchmark results can be made public or not.

A

Glossary

Tflop/s - Tera- floating point operations per second.

GUPS - Giga- updates per second.

tmpC - Transactions per minute. A benchmark rating defined by TPC (Transaction Processing Performance Council) that measures the overall transaction processing performance of a system.

QppH - "The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric (QphH@Size), and reflects multiple aspects of the capability of the system to process queries.

DOM - Document Object Model. *A programming API for HTML and XML documents*[27]

GUI - Graphical User Interface. A user-interface that contains different visual elements and the user can interact with these elements

B

SIG Score

B.1. First Upload

De code van het systeem scoort 3,5 ster op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Duplication en Unit Size.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er bijvoorbeeld duplicatie te vinden tussen GraphalyticsCore.py en /GraphalyticsDriver.py. Het is in dit geval beter om de gedeelde code naar een parent class te verplaatsen, en beide classes vervolgens deze parent te laten extenden. Zo voorkom je dat je toekomstige aanpassingen dubbel hoeft te doen.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. In jullie project is compute_score() in GraphalyticsCompetition.py een voorbeeld van een methode die beter leesbaar gemaakt zou kunnen worden. Je zou het deel waarbij je de "curr_alg_p_instances" lijst vult (regel 59-65) bijvoorbeeld kunnen uitsplitsen naar een aparte methode. Hetzelfde principe geldt voor het opbouwen van de "dataset_dict".

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de testcode ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

– Dennis Bijlsma, Software Improvement Group

B.2. Second Upload

In de tweede upload zien we dat het codevolume is gestegen, terwijl de score voor onderhoudbaarheid ongeveer gelijk is gebleven.

Qua duplicatie zien we dat de eerder genoemde voorbeelden zijn aangepast. De score gaat echter niet omhoog, omdat de verbeteringen weer grotendeels ongedaan zijn gemaakt doordat in de nieuwe code ook weer nieuwe duplicatie is toegevoegd. Zo wordt in het bestand AdminSubmissionTableGenerator.ts bijvoorbeeld steeds hetzelfde stuk configuratie herhaald. Het is beter om zoiets naar een aparte utility-methode te verplaatsen en die vervolgens aan te roepen. Ook zien we opnieuw duplicatie tussen twee soortgelijke Python-bestanden, zoals bijvoorbeeld GraphalyticsCompetition.py en ScalingCompetition.py

Wel is het positief dat jullie naast nieuwe productiecode ook nieuwe testcode hebben toegevoegd. De groei van de testcode blijft nog wel wat achter, kijk uit dat je op termijn niet een situatie komt waarbij sommige delen van de code heel goed worden getest en andere delen bijna niet.

Uit deze observaties kunnen we concluderen dat de aanbevelingen op de eerste upload deels zijn meegenomen in het vervolg van het ontwikkeltraject.

– Dennis Bijlsma, Software Improvement Group

C

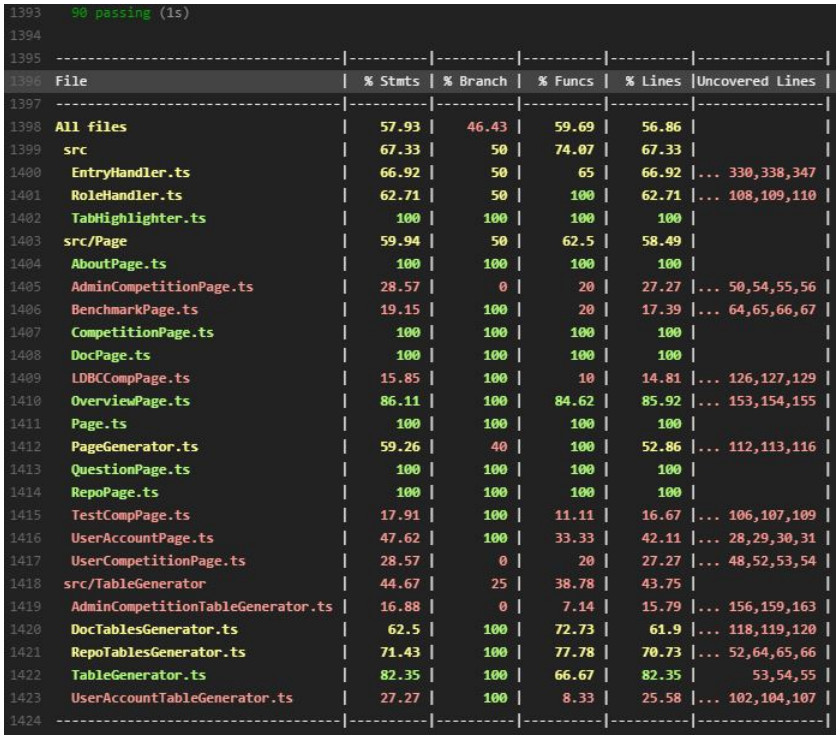
Front-end Test Coverage

```
  47 passing (660ms)

----------------------|----------|----------|----------|----------|----------------|
File                  |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------------|----------|----------|----------|----------|----------------|
All files             |      100 |      100 |      100 |      100 |                |
 src                  |      100 |      100 |      100 |      100 |                |
  TabHighlighter.ts   |      100 |      100 |      100 |      100 |                |
 src/Page             |      100 |      100 |      100 |      100 |                |
  AboutPage.ts        |      100 |      100 |      100 |      100 |                |
  CompetitionPage.ts  |      100 |      100 |      100 |      100 |                |
  DocPage.ts          |      100 |      100 |      100 |      100 |                |
  OverviewPage.ts     |      100 |      100 |      100 |      100 |                |
  Page.ts             |      100 |      100 |      100 |      100 |                |
  QuestionPage.ts     |      100 |      100 |      100 |      100 |                |
  RepoPage.ts         |      100 |      100 |      100 |      100 |                |
----------------------|----------|----------|----------|----------|----------------|
```

Figure C.1: Test coverage of the front-end in the first few weeks (image is taken from Travis build log)

```
  92 passing (2s)

--------------------------|----------|----------|----------|----------|----------------|
File                      |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
--------------------------|----------|----------|----------|----------|----------------|
All files                 |    83.66 |    68.89 |    81.68 |    83.23 |                |
 src                      |    73.74 |       50 |    73.08 |    73.74 |                |
  EntryHandler.ts         |    66.92 |       50 |       65 |    66.92 |... 330,338,347 |
  RoleHandler.ts          |    92.11 |       50 |      100 |    92.11 |       20,21,22 |
  TabHighlighter.ts       |      100 |      100 |      100 |      100 |                |
 src/Page                 |     94.4 |    95.24 |    92.86 |    94.15 |                |
  AboutPage.ts            |      100 |      100 |      100 |      100 |                |
  CompetitionPage.ts      |      100 |      100 |      100 |      100 |                |
  DocPage.ts              |      100 |      100 |      100 |      100 |                |
  OverviewPage.ts         |    86.49 |      100 |    84.62 |    86.11 |... 163,164,165 |
  Page.ts                 |      100 |      100 |      100 |      100 |                |
  PageGenerator.ts        |    97.78 |    88.89 |      100 |     97.5 |             72 |
  QuestionPage.ts         |    79.07 |      100 |    77.78 |    78.57 |... 65,66,79,80 |
  RepoPage.ts             |      100 |      100 |      100 |      100 |                |
 src/TableGenerator       |     67.2 |    33.33 |    73.91 |    66.67 |                |
  DocTablesGenerator.ts   |    60.61 |    33.33 |    72.73 |       60 |... 123,124,125 |
  RepoTablesGenerator.ts  |    71.43 |      100 |    77.78 |    70.73 |... 52,64,65,66 |
  TableGenerator.ts       |    82.35 |      100 |    66.67 |    82.35 |       53,54,55 |
--------------------------|----------|----------|----------|----------|----------------|
```

Figure C.2: Test coverage of the front-end before we adjusted the planning (image is taken from Travis build log)



```
  90 passing (1s)

-----------------------------------|----------|----------|----------|----------|----------------|
File                               |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------------------------|----------|----------|----------|----------|----------------|
All files                          |    57.93 |    46.43 |    59.69 |    56.86 |                |
 src                               |    67.33 |       50 |    74.07 |    67.33 |                |
  EntryHandler.ts                  |    66.92 |       50 |       65 |    66.92 |... 330,338,347 |
  RoleHandler.ts                   |    62.71 |       50 |      100 |    62.71 |... 108,109,110 |
  TabHighlighter.ts                |      100 |      100 |      100 |      100 |                |
 src/Page                          |    59.94 |       50 |     62.5 |    58.49 |                |
  AboutPage.ts                     |      100 |      100 |      100 |      100 |                |
  AdminCompetitionPage.ts          |    28.57 |        0 |       20 |    27.27 |... 50,54,55,56 |
  BenchmarkPage.ts                 |    19.15 |      100 |       20 |    17.39 |... 64,65,66,67 |
  CompetitionPage.ts               |      100 |      100 |      100 |      100 |                |
  DocPage.ts                       |      100 |      100 |      100 |      100 |                |
  LDBCCompPage.ts                  |    15.85 |      100 |       10 |    14.81 |... 126,127,129 |
  OverviewPage.ts                  |    86.11 |      100 |    84.62 |    85.92 |... 153,154,155 |
  Page.ts                          |      100 |      100 |      100 |      100 |                |
  PageGenerator.ts                 |    59.26 |       40 |      100 |    52.86 |... 112,113,116 |
  QuestionPage.ts                  |      100 |      100 |      100 |      100 |                |
  RepoPage.ts                      |      100 |      100 |      100 |      100 |                |
  TestCompPage.ts                  |    17.91 |      100 |    11.11 |    16.67 |... 106,107,109 |
  UserAccountPage.ts               |    47.62 |      100 |    33.33 |    42.11 |... 28,29,30,31 |
  UserCompetitionPage.ts           |    28.57 |        0 |       20 |    27.27 |... 48,52,53,54 |
 src/TableGenerator                |    44.67 |       25 |    38.78 |    43.75 |                |
  AdminCompetitionTableGenerator.ts|    16.88 |        0 |     7.14 |    15.79 |... 156,159,163 |
  DocTablesGenerator.ts            |     62.5 |      100 |    72.73 |     61.9 |... 118,119,120 |
  RepoTablesGenerator.ts           |    71.43 |      100 |    77.78 |    70.73 |... 52,64,65,66 |
  TableGenerator.ts                |    82.35 |      100 |    66.67 |    82.35 |       53,54,55 |
  UserAccountTableGenerator.ts     |    27.27 |      100 |     8.33 |    25.58 |... 102,104,107 |
-----------------------------------|----------|----------|----------|----------|----------------|
```

Figure C.3: Test coverage of the front-end a few days after we adjusted the planning (image is taken from Travis build log)

Bibliography

[1] F. Amigoni, E. Bastianelli, J. Berghofer, A. Bonarini, G. Fontana, N. Hochgeschwender, L. Iocchi, G. Kraetzschmar, P. Lima, M. Matteucci, P. Miraldo, D. Nardi, and V. Schiaffonati. Competitions for benchmarking: Task and functionality scoring complete performance assessment. *IEEE Robotics Automation Magazine*, 22(3):53–61, Sept 2015. ISSN 1070-9932. doi: 10.1109/MRA.2015.2448871.

[2] C. Bach. jQuery plugin: Tablersorter 2.0. URL http://tablesorter.com/docs/.

[3] Bootstrap. Bootstrap - the world's most popular mobile-first and responsive front-end framework. URL http://getbootstrap.com/.

[4] chaijs.com. Chai. URL http://chaijs.com/.

[5] Sinon.JS committers. Standalone test spies, stubs and mocks for JavaScript. Works with any unit testing framework. URL http://sinonjs.org/.

[6] R. S. Cruz. rstacruz/jsdom-global: Enable DOM in Node.js. URL https://github.com/rstacruz/jsdom-global.

[7] Graph500. The graph500 list. URL http://www.graph500.org/results_nov_2016.

[8] HPCC. The HPCC challenge. URL http://www.tpc.org/tpcc/results/tpcc_results.asp?orderby=hardware.

[9] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. Prat-Pérez, Manhardt T., S. Depner, H. Chafi, Capota M., Sundaram N., Anderson M., et al. LDBC Graphalytics Benchmark v0.2.6 - Draft Release. URL https://github.com/ldbc/ldbc_graphalytics_docs/blob/master/LDBC-Graphalytics_tech-specs_v0.2.6.pdf.

[10] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. Prat-Pérez, T. Manhardt, H. Chafi, M. Capota, N. Sundaram, M. Anderson, et al. Ldbc graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *Proceedings of the VLDB Endowment*, 9(13), 2016.

[11] Swen Jacobs, Roderick Bloem, Romain Brenguier, Robert Könighofer, Guillermo A Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, et al. The second reactive synthesis competition (syntcomp 2015). *arXiv preprint arXiv:1602.01171*, 2016.

[12] Logilab. Pylint | code analysis for Python | www.pyling.org. URL https://www.pylint.org/.

[13] Microsoft. TypeScript - JavaScript that scales. URL https://www.typescriptlang.org/.

[14] mochajs.org. Mocha - the fun, simple, flexible javascript test framework. URL https://mochajs.org/.

[15] Node.js. Node.js. URL https://nodejs.org/en/about/.

[16] npm inc. npm. URL https://www.npmjs.com/.

[17] pythonhosted.org. flask-mail — Flask-Mail 0.9.1 documentation. URL https://pythonhosted.org/Flask-Mail/.

[18] A. Ronacher. Welcome | Flask (A Python Microframework). URL http://flask.pocoo.org/.

[19] Python Software Foundation. Flask-Bcrypt 0.7.1 : Python Package Index, . URL https://pypi.python.org/pypi/Flask-Bcrypt.

[20] Python Software Foundation. pep8 1.7.0: Python Package Index, . URL https://pypi.python.org/pypi/pep8.

[21] SPEC. Standard performance evaluation corporation. URL http://www.tpc.org/tpcc/results/tpcc_results.asp?orderby=hardware.

[22] Palantir Technologies. TSlint - an extensible linter for the typeScript language. URL https://palantir.github.io/tslint/.

[23] Top500. Top500 supercomputer site. URL https://www.top500.org/lists/2016/06/.

[24] TPC. TPC and TPC-H. URL http://www.tpc.org/tpcc/results/tpcc_results.asp?orderby=hardware.

[25] GmbH Travis CI. Travis-CI - Test and Deploy Your Code with Confidence. URL https://travis-ci.com/.

[26] K. Waters. Prioritization using MoSCoW, 2009. URL http://www.allaboutagile.com/prioritization-using-moscow/.

[27] World Wide Web Consortium. What is the Document Object Model? URL https://www.w3.org/TR/WD-DOM/introduction.html.