NTNU
Norwegian University of
Science and Technology

Improving System Usability of Climbing Mont Blanc - An Online Judge for Energy Efficient Programming

Sindre Magnussen

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Lasse Natvig, IDI
Co-supervisor: Magnus Själander, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Climbing Mont Blanc - Improving System Usability

Climbing Mont Blanc (CMB) is a system for evaluation of programs executed on modern heterogeneous multicores such as the Exynos Octa chips used in eg. Samsung Galaxy S5 and S6 mobile phones, see https://www.ntnu.edu/idi/card/cmb. CMB evaluates both performance and energy efficiency, and provides the possibility of performance ranking lists and online competitions. A first version of the system is available and under trial use. This master thesis project builds on the project work by Sindre Magnussen finished in December 2015, and is focusing at continuing the improvement of various aspects of the system and its use.

The project involves the following subtasks:

1. Fix the main bugs and known issues found during user testing of CMB in November 2015.
2. Change and/or optimize the existing database management system if necessary to handle more frequent users submission.
3. Improve and extend the CMB system's usability features in accordance with the CMB team's priorities.
4. Conduct a user-experiment to evaluate usability.

If time permits:

5. Propose improvements to the existing stability test with a practical solution for simulating users and their submissions, i.e. a synthetic workload.
6. Propose how to improve the how-to information and the existing database of problems by cleaning up, improving, using experience from TDT-4200 and adding new problems.
7. Propose how to implement a discussion forum which allows discussion of each problem and the use of CMB in general.
8. Implement some of the proposed solutions after approval by, and in collaboration with the CMB team.

The master thesis project is part of the EECS Strategic Research project at IME (www.ntnu.edu/ime/eecs). Many of the tasks assume a good collaboration with master student Christian Chavez.

This master thesis project is reserved for master student Sindre Magnussen.

Main supervisor: Prof. Lasse Natvig
Co-supervisor: Assoc. prof Magnus Själander

Abstract

For each release of a new smartphone model, the limits of their CPUs, so called heterogeneous multicore processors, are pushed. As a result, the usage of such processors has gained an increased interest outside the mobile market and they are now candidates for building energy efficient supercomputer architectures. The European Mont-Blanc project has for its objective to develop a high-performance supercomputer architecture, using smartphone processors, to lower energy consumption by 15 to 30 times compared to other high-performance supercomputers. The project has resulted in multiple prototypes using heterogeneous processors, and has shown promising results for future development of energy efficient supercomputer architectures.

Regarding energy efficiency, there are enormous challenges ahead for both hardware and software developers. To aid developers to create energy efficient software, Lasse Natvig started the Climbing Mont Blanc project. Inspired by other online programming competition systems, or Online Judge systems, the idea is to let programmers compete to develop energy efficient code to various programming problems. Climbing Mont Blanc is, to our knowledge, the first Online Judge focusing on energy efficiency of submitted code. However, previous user feedback has identified some components which need improved usability.

This thesis looks into improving certain usability aspects of components in the Climbing Mont Blanc system. Real-time notifications, updated feedback messages, a bulletin board, an updated user interface, and fixes of known issues have been integrated into version two of the Climbing Mont Blanc judge. A user experiment has shown that users are more satisfied with the feedback given in system version two, by a confidence value of 97.2%. There is also a noticeable trend that the users are more satisfied with the design and the provided how-to information of the new system version. Furthermore, system usability has been continuously validated by conducting small informal user tests throughout the work on the project.

Sammendrag

Grensene til prosessorer brukt i smarttelefoner, såkalte heterogene multikjerne prosessorer, flyttes hver gang en ny smartelefonmodell lanseres. Dette har ført til økt interesse utenfor mobilmarkedet for denne typen prossesorer, og de har vist seg å være gode kandidater for å bygge energieffektive superdatamaskinarkitekturer. Det Europeiske prosjektet Mont Blanc har som mål å bruke de nevnte mobilprosessorene til å utvikle en superdatamaskin, som konsumerer 15 til 30 ganger mindre energi sammenlignet med tilsvarende arkitekturer. Prosjektet har underveis resultert i flere prototyper som bruker heterogene multikjerne prosessorer, og prototypene har vist lovende resultater for videre utvikling av energieffektive superdatamaskiner.

Det er flere utfordringer knyttet til energieffektivitet i disse systemene som utfordrer både maskinvare- og systemutviklere. For å hjelpe utviklere å lage energieffektive løsninger, startet Lasse Natvig Climbing Mont Blanc-prosjektet. Systemet er inspirert av andre nettbaserte systemer som tilbyr programmeringskonkurranser, såkalte nettbaserte dommere ("Online Judges"). Tanken bak systemet er å la programmerere konkurrere i å utvikle energieffektive løsninger til et bredt spekter av programmeringsoppgaver. Climbing Mont Blanc-systemet er såvidt vi vet det eneste systemet som fokuserer på energieffektiv koding. Tilbakemeldinger fra brukere viser derimot at noen deler av systemet trenger å forbedre brukbarheten.

Denne avhandlingen ser på hvordan brukbarheten til visse deler av systemet kan forbedres. Sanntidsnotifikasjoner, oppdateringer av feedback meldinger, en elektronisk oppslagstavle, oppdateringer av brukergrensesnittet og fjerning av kjente feil har blitt integrert inn i versjon to av Climbing Mont Blanc-systemet. Et brukereksperiment påviste at brukere er mer fornøyd med feedback gitt av system versjon to, med en konfidensverdi på 97.2%. Resultatet av eksperimentet viste også en tendens til at brukere var mer fornøyd med designet og tilgjengelig bruksinformasjon i system versjon to. Kontinuerlige lavterskel brukertester har også blitt gjennomført for a validere brukbarheten til systemet underveis i utviklingen.

Preface

This thesis is submitted to fulfill the remaining requirements for an MSc degree in computer science at Norwegian University of Science and Technlogy (NTNU), Trondheim. This work has been conducted at the Department of Computer and Information Science NTNU throughout the spring of 2016, alongside being employed as part of the course staff as a teaching assistant in TDT4102 [TDTa].

Acknowledgements

I would like to thank my supervisors Lasse Natvig and Magnus Själander for accurate and valuable feedback on this work. I would also like to especially thank Lasse Natvig for letting me contribute to the Climbing Mont Blanc project.

Thank you, Guttorm Sindre for all your help when planning the user experiment and your help with analyzing the experiment results.

Thank you, Thea Mathisen and Johannes Lier for all your help and feedback, which has been really valuable to me.

Thank you, Arne-Dag Fidjestøl for helping resolve maintenance problems at the CMB servers.

Thank you, Jørn Eriksen for providing feedback on clarity of the produced text for an external professional.

Finally, I would like to thank Øystein Kvamme Repp, Håkon Åmdal, and others at the study hall Sule for providing feedback on drafts of this thesis.

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface.
CA	Certificate Authority.
CI	Continuous Integration.
CLI	Command Line Interface.
CMB	Climbing Mont Blanc.
DBMS	Database Management System.
EDP	Energy-Delay Product.
FIFO	First-In-First-Out.
FLOPS	Floating Point Operations per Second.
HPC	High-Performance Computing.
HTTP	Hyper Text Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
ICPC	International Collegiate Programming Contest.
IE	Internet Explorer.
ISA	Instruction Set Architecture.
JSON	JavaScript Object Notation.
MB	Mont Blanc.
MVC	Model-View-Controller.
OJ	Online Judge.
ORM	Object Relational Mapper.
OS	Operating System.
REST	Representional State Transfer.
SSE	Server-Sent Events.

SSH	Secure Shell.
SSL	Secure Socket Layer.
UFW	Uncomplicated FireWall.
WMW	Wilcoxon-Mann-Whitney.

Chapter 1

Introduction

1.1 Motivation

For each new smartphone model, the limits of smartphone processors regarding computational power and energy efficiency are pushed. As a result, processors for mobile devices, or so-called heterogeneous multicores, have gained interest as components in systems outside the mobile market. The Mont Blanc (MB) Project [MB] has shown that heterogeneous multicores are potential candidates for building High-Performance Computing (HPC) systems, as the performance gap between heterogeneous multicores and microprocessors is closing quickly [RCG$^+$13]. Increasing performance has always been a priority in HPC, and HPC system performance is expected to hit Exascale performance (10^{18} Floating Point Operations per Second (FLOPS) (exaFLOPS)) within few years [TOPa]. The main goal of the MB project is to build a new type of supercomputer architecture, reaching Exascale performance using 15 to 30 times less energy compared to other HPC systems. The project has already constructed multiple prototypes by using heterogeneous processors like the Samsung Exynos [EXY] processor among others. The Barcelona Supercomputing Center coordinates the project, and the project is funded until September 2016.

Making energy efficient systems poses enormous challenges for both hardware and software developers. However, ARM among others have made progress in producing energy efficient hardware. ARM has developed the energy aware big.LITTLE technology for their heterogeneous cores, which consists of both high performance and energy efficient processors. The technology uses Global Task Scheduling to assign threads to the most appropriate CPU based on dynamic runtime information [Jef13]. Other well-known hardware techniques are also massively used, such as Dynamic Voltage and Frequency Scaling. Among development of energy efficient software, Task-Based Programming models like OmpSs [PBAL13] have gained increasing interest in recent time, and have been used to develop task based programs for heterogeneous architectures [LNAM12]. Both research areas are expected to become even more important in the future.

There will be an increased need for programmers skilled in developing energy efficient code for heterogeneous multicores. Many Online Judges (OJs) such as PKU Online Judge [PKU] and Kattis [KAT] has existed for years, making it possible for programmers to compete, and self-educate in creating efficient code. However, none of the OJs we are aware of are known to evaluate and focus on energy efficiency of submitted programs. Noticing the need for such an OJ, Lasse Natvig started the Climbing Mont Blanc (CMB) project to train programmers in developing energy efficient code for heterogeneous multicores by providing a new OJ. Torbjørn Follan and Simen Støa developed the first prototype version during the Spring of 2015 [FS15]. Previous user testing conducted during the specialization project[1] has, however, shown that usability improvements are needed in some of the system components.

The CMB project wants to stimulate students and programmers to develop energy efficient high-performance code for heterogeneous multicores. The PKU Online Judge [PKU] had almost a total of 1.3 million submissions last year, and we believe that increasing the number of submissions on the CMB system might help to solve some of the challenges related to programming heterogeneous multicores. An important factor in motivating use of the system, is believed to be achieved by improving system usability. This thesis aims to further improve the system and extend with more functionality with a focus on usability.

1.2 Problem Statement Interpretation

Three different types of objectives are identified in the problem statement. Usability improvements are the main focus of this thesis and there are several tasks listed in the problem statement that can be interpreted as usability tasks. Further, there are tasks not strictly related to usability that can instead be viewed as general improvements to the system. Finally, the thesis aims to propose a series of improvements and implement them if time permits. The subtasks listed below will, as of the above discussion, either be labeled as **U** for usability, **I** for improvements, or **P** for improvement proposals. As the tasks were already prioritized by the supervisor in the problem statement, the tasks listed in secondary objectives are assumed to be less important compared to the main objectives. However, they are of importance to the CMB project in general.

Main objectives

U1 Fix the main bugs and known issues found during user testing of CMB in November 2015:

[1]The earlier project work mentioned in the problemstatement.

- **U1.1 Uploads on Mac OS X**: Do not work and give an uninformative error message.
- **U1.2 Submissions locked in a running state**: There should be a timeout on submissions containing infinite loops or code which may leave the submission in a inconsistent state.
- **U1.3 Highscore list bug**: When switching the sort metric, private submissions should still be hidden.

I1 Change and/or optimize the existing Database Management System (DBMS) if necessary to handle more frequent user submissions. Early in this thesis we discovered that optimizing the database was not as important at this point as first predicted, as the system is still a prototype and we have few active users on the system. As a result, we did not think that optimizing the database was of importance at this point. However, switching to a more sophisticated DBMS should still be done.

U2 Improve and extend the CMB system's usability features in accordance with the CMB team's priorities. The features prioritized by the CMB team is a combination of features found during the specialization project, previous user feedback, and necessary usability features discovered throughout the semester:

- **U2.1 Improved feedback during errors**: Error messages should be structured, clear, and visible.
- **U2.2 Add support for real-time updates during submission execution**: State of submissions should update in real-time without the need of manually refreshing web-content to check for state changes.
- **U2.3 Newly added problems should be more stable**: Problem files added through the admin interface sometimes behaved strange during the semester, such that the success of submissions on the problem seemed random even with a correct implementation.
- **U2.4 Bulletin board extension**: A bulletin board is needed to display administrator messages to the CMB users.
- **U2.5 Views upgrade**: Views should only display necessary information to users. It is also important that users are satisfied with the judge's user interface.

U3 Conduct a user-experiment to evaluate system usability. Since a user test were conducted in the specialization project, the results of the previous test is used for comparison to evaluate the work done in this thesis.

Secondary objectives

P1 Propose improvements to the existing stability test with a practical solution for simulating users and their submissions, i.e., a synthetic workload.

P2 Propose changes to the HowTo information, and how to improve the existing database of problems by cleaning up and improving problem descriptions using experience gained during the system's use in TDT4200 [TDTb]. Add new problems if time permits.

P3 Propose how to implement a discussion forum which allows discussion of each problem and the use of CMB in general.

I2 Implement some of the proposed solutions after approval by, and in collaboration with the CMB team.

1.3 Project Contributions

This project contributes towards improving the usability of the Climbing Mont Blanc system. We define usability as the ease of use, ease of learning, satisfaction of a software system, as well as user error rates. Chapter 3 presents the formal definition of usability used in this thesis. This thesis also assumes that the usability of a software system is extended if more features are added to the system frontend, since the range of possible actions for users is extended. Usability is a broad term and there is always room for usability improvements in a software system. This thesis is restricted to improve a subset of prioritized usability aspects prior to its the start, as well as urgent usability aspects discovered throughout the work on this thesis.

Users of the system is defined as both administrators using the admin interface, and regular users using the frontend browser interface (the website `https://climb.idi.ntnu.no`). Furthermore, this thesis also aims to further extend the functionality of the system with focus on contributing towards usability. All contributions and their corresponding sections are summarized in Table 1.1.

It is also worth mentioning that contributions have been made to the CMB project, which are not related to the objectives set in Section 1.2. A rough estimate of said contributions is calculated to be around 15%; about 5% of the time was spent on helping other Master students with development or usage of the system, and 10% was spent on maintaining the system, upgrading code libraries and packages, as well as attending meetings to plan the future of the CMB project. These contributions are not discussed in this work.

Objective	**Description Summary**	**Section(s)**
`U1`	Fix the main bugs and known issues found during previous user testing of CMB.	Section 4.2.1 and 4.4
`I1`	Switch the DBMS system used by the CMB system.	Section 4.3.1
`U2`	Improve or extend the CMB system's usability features.	Section 4.1, 4.2.2, and 4.3.2
`U3`	Conduct a user-experiment to evaluate system usability.	Section 5.1 and 5.2
`P1`	Propose improvements to the existing stability test.	Section 4.5.1
`P2`	Propose improvements to the HowTo-page and existing problem descriptions, and add new problems.	Section 4.5.2 and 4.5.3
`P3`	Propose how to implement a discussion forum.	Section 4.5.4

Table 1.1: Thesis contributions

1.4 Outline

This report is structured as follows:

Chapter 2: The chapter presents the Mont Blanc project, which motivated the Climbing Mont Blanc project. The chapter will also present the Climbing Mont Blanc project, that is, the system architecture, energy measurement method, the system environment, and system security. The chapter will also present the theme of another Master Thesis working to improve system scalability. Finally, related OJs and other related work concludes the chapter.

Chapter 3: The chapter presents the definition of usability in software systems. Based on theory, aspects which makes an OJ usable are briefly discussed. The chapter ends with a discussion of the objectives set by this thesis and their relevance to the usability definition.

Chapter 4: The chapter presents implementation done to the system in this work. The chapter will present the technical solutions for the usability goals set in Section 1.2, as well as present implementation proposals for a improved stability test, improved HowTo-information and base of provided problems, and a discussion forum.

Chapter 5: The chapter starts with a presentation of continuous user testing conducted during development. The chapter will also present a user experiment and analyze wether the usability has improved after implementing the usability features mentioned in Section 1.2. The chapter ends with a presentation of system unit test coverage, to justify the correctness of implemented features.

Chapter 6: An evaluation and discussion of the technologies used and user test pros and cons will be discussed in this chapter. The chapter also presents alternatives to technologies and the user test methodology. The chapter ends with a discussion on wether the objectives presented in Section 1.2 have been reached.

Chapter 7: The chapter concludes the thesis, and presents future work suggestions based on results of the user experiment and suggested improvements listed in Appendix C.

Chapter 2

Background

This chapter explains the background for this thesis. Section 2.1 introduces the Mont Blanc Project, the project goals, and its current status, as well as the largest and most significant prototypes. Section 2.2 presents the CMB project, the CMB system architecture, energy measurement procedures, code correctness and development tools, and other related CMB projects conducted during the Spring of 2016. The chapter finishes with a presentation of related work, presenting other OJs as well as relevant crowdsourcing web sites.

2.1 The Mont Blanc Project

The MB project [MB] started in October 2011. The project received 14 million Euros as initial funding, and the European Commission granted eight of those millions. The project received an additional funding of eight million Euros from the European Commission after two years. With a total funding of 22 million Euros, it was estimated that the project would last until September 2016. The project has also been extended as of October 2015, with an extended budget of 7.9 million Euros funded by the European Commission. As mentioned in Section 1.1, the long-term goal is to develop an HPC architecture with Exascale performance that uses 15 to 30 times less energy than other HPC systems. The energy performance metric used is FLOPS/W, motivated by the Green500 list [GRE]. The project is split into three phases; the first two are coordinated by the Barcelona Supercomputing Center [BSC], while Bull [BUL] coordinates the third phase.

2.1.1 Project Goals and Status

Both Ramirez [Ale14a] and Mantovani [Fil15] summarized the goals of the first two project phases. Mont-Blanc phase 1, valid from 2011 to 2015, had three main objectives. The first was to deploy a prototype scalable to 50 PFLOPS with a total power consumption of 7MW using the available energy efficient embedded technology, which should be competitive with the Green500 leaders of 2014. The second objective was to overcome some limitations found during development and use the gained knowledge in the design of the next generation HPC system. They aimed for an HPC architecture that should be scalable to 200 PFLOPS on 10 MW, which should be competitive with the Top500 leaders of 2017. The third and last objective was to port and optimize Exascale scientific applications capable of exploiting the new generation HPC systems described in objective 2.

The MB project phase 2 valid from 2013 to September 2016 has four objectives. The first objective is to supplement the Mont-Blanc prototype, presented below, software stack with software development tools and support ARMv8 Instruction Set Architecture (ISA). The second goal is to construct the initial definition of the Exascale architecture, which involves deployment of small ARM-based mini-clusters and evaluation of their suitability in HPC. Objective number three is being up-to-date with newly released ARM products and evaluate if they are suitable for HPC architectures. The last and fourth objective is continued support for the Mont Blanc project. These objectives are also summarized by Follan and Støa [FS15] and on the MB project website [MB].

Mantovani reported the project status in July 2015 [Fil15]. Among the tasks finished so far are the deployment of a software stack to support the Mont-Blanc system (see prototype specifications in Section 2.1.2), porting of new HPC kernels and applications, and design, development, deployment and monitoring of the Mont-Blanc prototype. Amidst the ongoing work is ARM 64 bit exploration, porting new applications for the HPC architecture developed, enhance the programming model (OmpSs [OMP]) used and monitor the Mont-Blanc prototype for fault tolerant techniques.

The third phase of the project started in October 2015. The goal is to design a new HPC platform that can improve the performance-energy ratio when executing actual applications. The new design will be developed using a hardware-software co-design [MG97] approach, and will ensure that hardware and system innovations are transformed into HPC application benefits.

2.1.2 Prototypes

Multiple prototypes have been announced, and the specifications of the most significant prototypes are listed below:

1. **Tibidabo** [RRP$^+$13]: Tibidabo is the first prototype of the MB project. The prototype is an experimental system and a proof-of-concept HPC system, to show that it is possible to deploy a large scale HPC cluster using ARM processors. The prototype contains compute-boards with NVIDIA Tegra 2 SoCs, with dual core ARM Cortex A9 @ 1GHz inside. The GPUs on the chip did not support the standard programming models OpenCL and CUDA and were therefore not used in the cluster. However, the compute cards can be replaced with up-to-date chips as they become available with updated GPUs

that support the two programming models. The prototype consists of 128 nodes each containing a Q7[1] module; each module holds a Tegra 2 chip. The prototype achieves 120 MFLOPS/W on High-Performance Linpack (HPL) benchmarks.

2. **Pedraforca** [Fil14]: Pedraforca consists of a combination of ARM processors and NVIDIA GPUs. One compute node includes a NVIDIA K20 GPU and a Tegra 3 Q7 Module containing 4 ARM Cortex A9 @ 1.3 GHz. The system includes 78 such compute nodes. It is the first large scale HPC system that makes use of ARM-based processors. Even though some factors of the initial technology stack limits the system, the prototype contributes towards the use of ARM-based embedded processors in HPC systems.

3. **Mont Blanc** [Ale14b, Fil15]: The latest prototype from the MB project. The system contains a rack with a total of 1080 Exynos 5 compute cards, or 2160 ARM Cortex-A15@1.7 GHz CPUs, 1080 ARM Mali T604 GPUs, 4.3 TB of RAM and 17.2 TB of Flash memory. The performance is about 35 TFLOPS and the power consumption is about 24 kW. In November 2014, the Mont Blanc Prototype had an energy efficiency of 1.5 GFLOPS/W. The best Green500 ranking during that time was approximately 5.2 GFLOPS/W.

2.2 The Climbing Mont Blanc Prototype

CMB is an OJ focusing on energy efficient programming on ARM based platforms, greatly inspired by the Mont Blanc project. An OJ is a web application or web based software system which allows its users to upload programming solutions to defined set of programming problems. A more formal definition of OJ systems can be found in Section 2.3. Most OJs serve problems with varying degrees of difficulty, and can compile and run multiple programming languages (see more in Section 2.3). The paper "*Climbing Mont Blanc - A Training Site for Energy Efficient Programming on Heterogeneous Multicore Processors*" [NFS+15] and the master thesis of Torbjørn Follan and Simen Støa [FS15] describes the system in great detail. The system version developed by Follan and Støa will hereby be referred to as *system version one*, while the updated system developed in this thesis will be refered to as *system version two*.

CMB is to the best of our knowledge the first OJ system that focuses on heterogeneous programming and measures energy consumption of submitted programs. The source code is held in a private repository on Bitbucket [BIT], that uses `git` [GIT] as version control. The architecture of CMB is shown in Figure 2.1. As seen in the figure, the system consists of three main parts; the *frontend*, the *server* and the *backend*. The three parts of the system are explained in detail below. Keep

[1]The specific module is taken out of production, but more information can be found in [RRP+13].

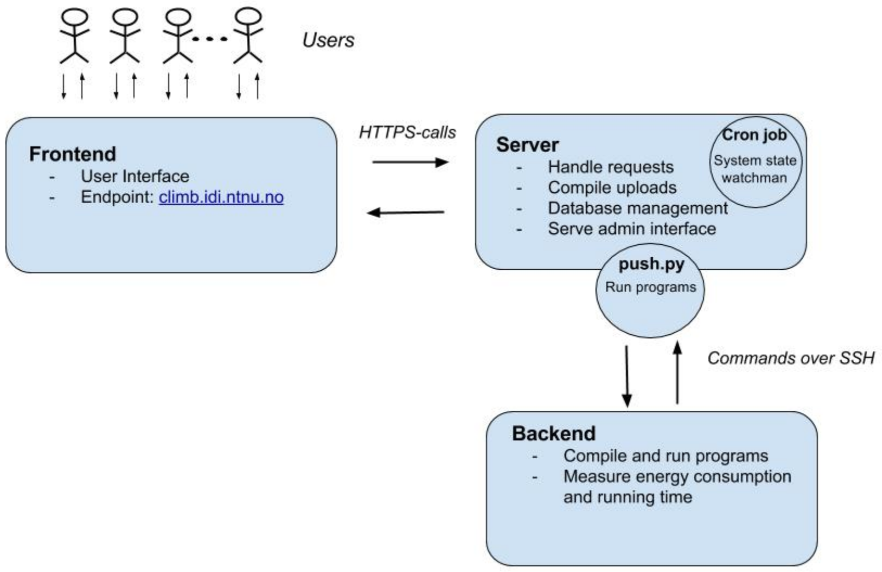


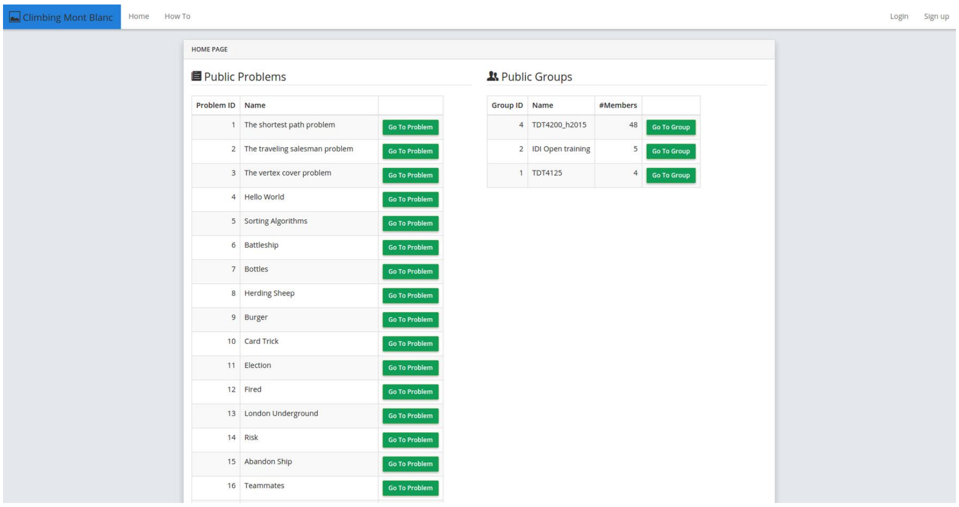
Figure 2.1: The Climbing Mont Blanc system architecture.

in mind that the below sub-sections present the prototype *before* any improvements were made to the system, i.e., system version one. Chapter 4 presents the implementations made in this thesis, i.e., the development of system version two.

2.2.1 Frontend

The frontend handles all user-based interaction. It is developed in AngularJS [ANG], which is a framework developed by Google and is maintained by Google and individual developers. Version one of the frontend uses Angular version 1.3 and Javascript ECMAScript 5th edition. Angular is based upon the Model-View-Controller (MVC) pattern [GHJV95] and extends HTML with dynamic views with two-way data binding for building single page applications. The result is a more pleasant user experience, as views are updated dynamically with model changes without the need of manually refreshing the web page. The framework also lets the developer define their own reusable components, which in most situations make the code base more structured. The view structure and styling are defined in HTML5 and CSS3 respectively. Google Analytics, an advanced monitoring service provided by Google, monitors user interaction at the frontend. Among the things monitored is the number of active users, number of new users, and user behavior.

The front page view of system version one is shown in Figure 2.2. Appendix B shows screenshots of version one and two of the system frontend. Many actions against the frontend, such as routing to a different view, launches Hyper Text



Public Problems

Problem ID	Name	
1	The shortest path problem	Go To Problem
2	The traveling salesman problem	Go To Problem
3	The vertex cover problem	Go To Problem
4	Hello World	Go To Problem
5	Sorting Algorithms	Go To Problem
6	Battleship	Go To Problem
7	Bottles	Go To Problem
8	Herding Sheep	Go To Problem
9	Burger	Go To Problem
10	Card Trick	Go To Problem
11	Election	Go To Problem
12	Fired	Go To Problem
13	London Underground	Go To Problem
14	Risk	Go To Problem
15	Abandon Ship	Go To Problem
16	Teammates	Go To Problem

Public Groups

Group ID	Name	#Members	
4	TDT4200_h2015	48	Go To Group
2	IDI Open training	5	Go To Group
1	TDT4125	4	Go To Group

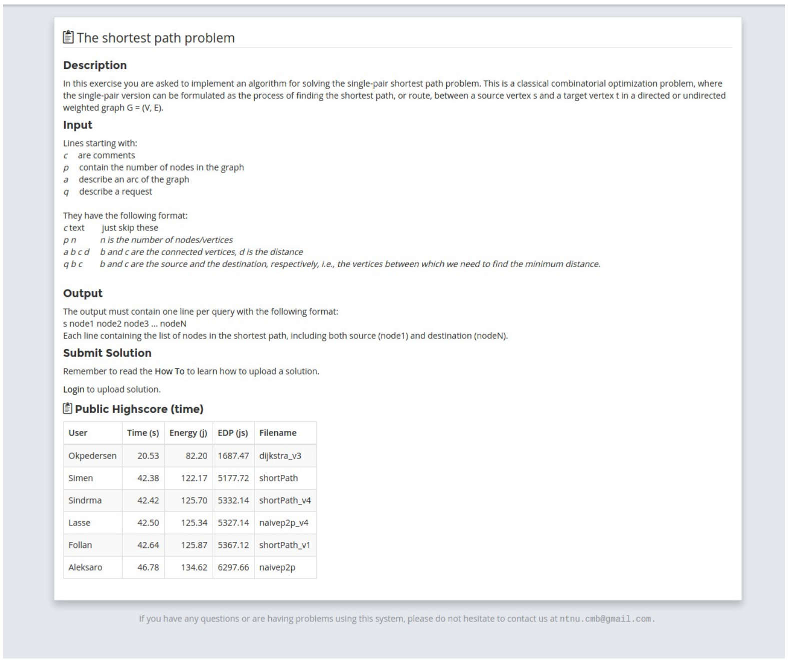
Figure 2.2: Climbing Mont Blanc Home Page.

Transfer Protocol (HTTP) requests to the server that changes models and updates the views dynamically. Angular controllers make these requests to fetch up-to-date data to the frontend models or to update the database with changes made to the models by the user. Routing to a particular problem is an example of dynamic updates of the view, and also displays different views depending on the user's state. If the user is not logged in, as shown in Figure 2.3a, the view only shows the submitted programs that are *visible*[2]. If the user then logs into the system, as shown in 2.3b, the problem view changes to make it possible to upload files and view previous submissions. Besides, many actions can be performed in the frontend, such as routing to a specific problem view, login, sign up, create and join groups, manage groups, view the HowTo-page, and see public problems. The most common action is to upload possible solutions to a problem, which requires a zipped folder containing all source files.

2.2.2 Server

The server is implemented as an Representional State Transfer (REST) Application Programming Interface (API), as defined in by Roy T. Fielding et.al [FT00]. The server is stateless, that is, the user state is stored at the frontend. It is also uniform, meaning that requests sent to and from the server use the same data format independent of the technologies used at the server and frontend. The RESTful API is implemented using Python Flask [FLAh] and the data format used is JavaScript Object Notation (JSON) [JSO]. Our development and production servers also make use of Gunicorn [GUN] to handle simultaneous requests from multiple users. Guni-

[2]A user may change the visibility of their submissions and results through the user interface.



The shortest path problem

Description

In this exercise you are asked to implement an algorithm for solving the single-pair shortest path problem. This is a classical combinatorial optimization problem, where the single-pair version can be formulated as the process of finding the shortest path, or route, between a source vertex s and a target vertex t in a directed or undirected weighted graph G = (V, E).

Input

Lines starting with:
c are comments
p contain the number of nodes in the graph
a describe an arc of the graph
q describe a request

They have the following format:
c text just skip these
p n *n is the number of nodes/vertices*
a b c d *b and c are the connected vertices, d is the distance*
q b c *b and c are the source and the destination, respectively, i.e., the vertices between which we need to find the minimum distance.*

Output

The output must contain one line per query with the following format:
s node1 node2 node3 ... nodeN
Each line containing the list of nodes in the shortest path, including both source (node1) and destination (nodeN).

Submit Solution

Remember to read the How To to learn how to upload a solution.

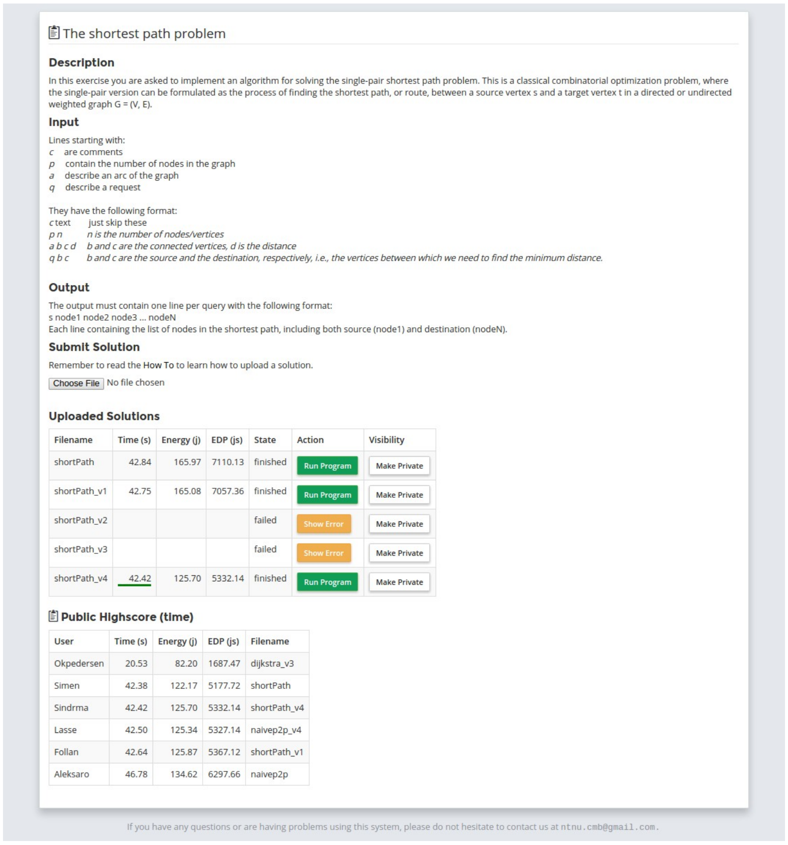
Login to upload solution.

Public Highscore (time)

User	Time (s)	Energy (j)	EDP (js)	Filename
Okpedersen	20.53	82.20	1687.47	dijkstra_v3
Simen	42.38	122.17	5177.72	shortPath
Sindrma	42.42	125.70	5332.14	shortPath_v4
Lasse	42.50	125.34	5327.14	naivep2p_v4
Follan	42.64	125.87	5367.12	shortPath_v1
Aleksaro	46.78	134.62	6297.66	naivep2p

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

(a) Problem View, logged out of CMB.



The shortest path problem

Description

In this exercise you are asked to implement an algorithm for solving the single-pair shortest path problem. This is a classical combinatorial optimization problem, where the single-pair version can be formulated as the process of finding the shortest path, or route, between a source vertex s and a target vertex t in a directed or undirected weighted graph G = (V, E).

Input

Lines starting with:
c are comments
p contain the number of nodes in the graph
a describe an arc of the graph
q describe a request

They have the following format:
c text just skip these
p n *n is the number of nodes/vertices*
a b c d *b and c are the connected vertices, d is the distance*
q b c *b and c are the source and the destination, respectively, i.e., the vertices between which we need to find the minimum distance.*

Output

The output must contain one line per query with the following format:
s node1 node2 node3 ... nodeN
Each line containing the list of nodes in the shortest path, including both source (node1) and destination (nodeN).

Submit Solution

Remember to read the How To to learn how to upload a solution.

Choose File No file chosen

Uploaded Solutions

Filename	Time (s)	Energy (j)	EDP (js)	State	Action	Visibility
shortPath	42.84	165.97	7110.13	finished	Run Program	Make Private
shortPath_v1	42.75	165.08	7057.36	finished	Run Program	Make Private
shortPath_v2				failed	Show Error	Make Private
shortPath_v3				failed	Show Error	Make Private
shortPath_v4	42.42	125.70	5332.14	finished	Run Program	Make Private

Public Highscore (time)

User	Time (s)	Energy (j)	EDP (js)	Filename
Okpedersen	20.53	82.20	1687.47	dijkstra_v3
Simen	42.38	122.17	5177.72	shortPath
Sindrma	42.42	125.70	5332.14	shortPath_v4
Lasse	42.50	125.34	5327.14	naivep2p_v4
Follan	42.64	125.87	5367.12	shortPath_v1
Aleksaro	46.78	134.62	6297.66	naivep2p

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

(b) Problem View, logged into CMB.

Figure 2.3: Climbing Mont Blanc Problem View States.

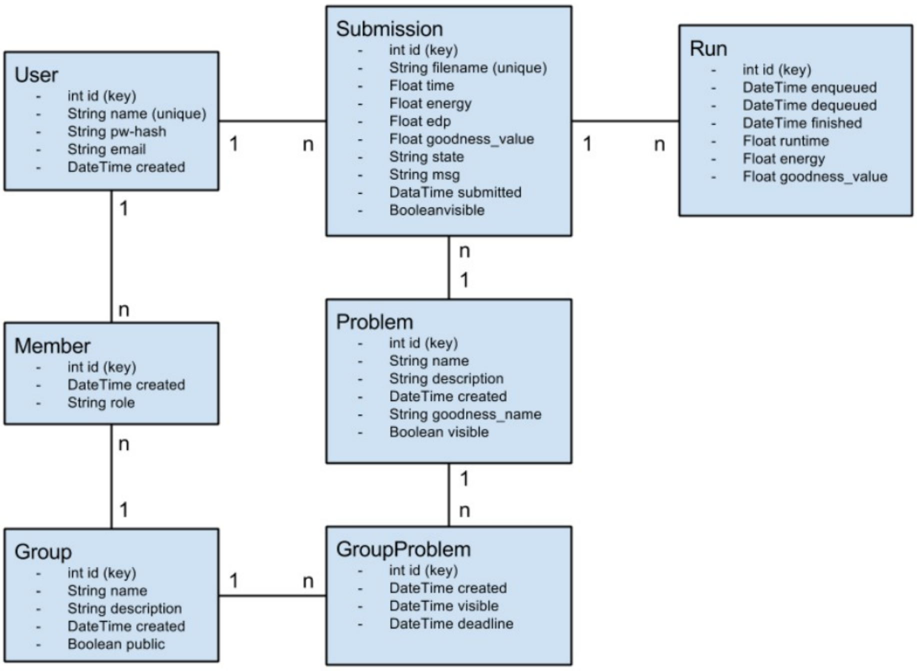


Figure 2.4: Database schema (taken from Master Thesis of Follan and Støa [FS15]).

corn forks off multiple workers upon server startup, each having a private instance of the Flask webserver, and routes incoming requests to the current available workers. If no workers are available, the requests are stalled until a worker becomes idle.

Nginx [NGI] is used as a *reverse proxy* on our development and production server. A reverse proxy serves static files on the fly while requests for dynamic content[3] are forwarded to Gunicorn. SQLite [SQLb] is used as database with SQLAlchemy [SQLa] on top. SQLAlchemy functions as an Object Relational Mapper (ORM), that is, it associates Python classes with database tables and objects of those classes with rows in the tables. SQLAlchemy makes it easy for the programmer to interact with the database, since SQL-statements are abstracted into Python objects and procedures. The underlying database schema is shown in Figure 2.4.

The server runs within a Python *Virtual Environment* [VIR] that have all required Python package dependencies[4] installed. The virtual environment contains only the packages needed by the server, which removes potential dependency errors if

[3]Dynamic content is data that changes during runtime, such as database content.

[4]The use of *dependencies* here refers to the dependencies between packages (e.g., code libraries or code APIs). As an example, SQLAlchemy might require a specific version of Python-Flask installed in order to work correctly.

the server is to host other Python-based systems in the future. The server is responsible for handling requests from the client, storing useful information in the database and filesystem, and compiling submitted programs.

A Python program called `push.py` runs in the background, regularly checking a First-In-First-Out (FIFO) queue for runnable programs. If a program is ready to run, the script will push the program from the server queue to the backend, and then compile and run it using Secure Shell (SSH). The result is returned to the server when execution terminates, and the server checks the correctness of the program before reporting the result back to the frontend. Furthermore, the server is responsible for monitoring the state of the system with a background cron job. The cron job runs every 15 minutes, checking if the three system parts Gunicorn, `push.py` and backend is up and running. The system sends out an automatic e-mail to the system administrators in case of system failure.

The server also provides an administrator interface. Admin users have special privileges which grant them access to the admin interface found at `http://climb.idi.ntnu.no/admin`. Procedures that modify the database or the server file system are all done through the admin interface. The admins control a lot of the content visible on the frontend, such as programming problems visibility. The most common administrator operation is to add new problems, which requires the admin to add a new problem to the database, and upload a set of files to the server used validate submitted code.

The files needed to make a problem solvable are an input file to be piped into the submitted programs, a file that holds the expected result of an execution, and a special file called `checker.cpp` that checks the correctness of submitted programs. When a program is executed, the input files are given as arguments to the submitted program. When the program is done executing, the checker compares the output of the submitted program against the expected answer files. The program is accepted if the checker program approves the program output. Since the checker is a regular C++-program, it is up to the admin to define what sort of output that should be accepted by the checker. Most of the problems available simply check the difference between output and expected output using the Unix `diff` program [DIF]. A special database field called *goodness* can also be added to the problem. The database field is meant for approximation problems to check how "good" a solution is. As an example, a solution to the Vertex Cover problem might yield a different cover on each run. The admin who created the problem can then define in the checker how good a given cover size is. More information on how to add problems through the admin interface can be found in Appendix D.

2.2.3 Backend

The backend is the executing unit of the CMB system. It is an Odroid XU3 board [XU3a], which consists of an Exynos 5 Octa heterogeneous multicore. A block diagram of the board is shown in Figure 2.5. The Odroid board uses a minimum

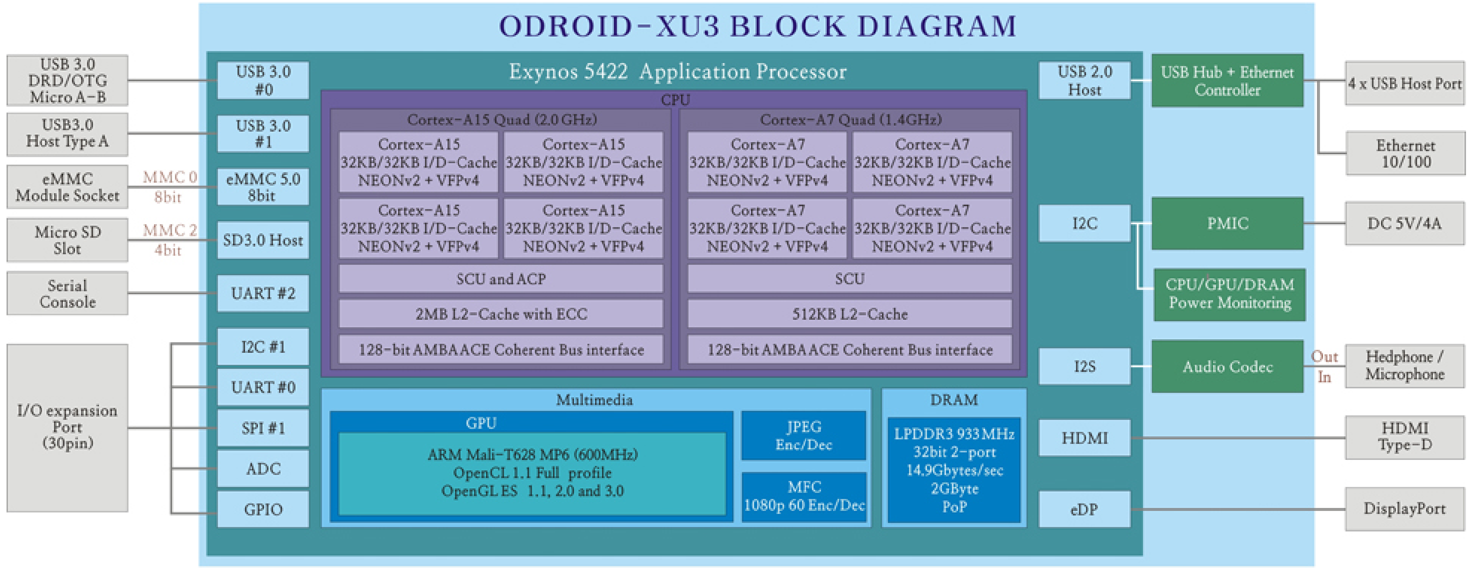

ODROID-XU3 BLOCK DIAGRAM
Exynos 5422 Application Processor
CPU
Cortex-A15 Quad (2.0 GHz)
Cortex-A15 32KB/32KB I/D-Cache NEONv2 + VFPv4
SCU and ACP
2MB L2-Cache with ECC
128-bit AMBA ACE Coherent Bus interface
Cortex-A7 Quad (1.4GHz)
Cortex-A7 32KB/32KB I/D-Cache NEONv2 + VFPv4
SCU
512KB L2-Cache
128-bit AMBA ACE Coherent Bus interface
Multimedia
GPU
ARM Mali-T628 MP6 (600MHz) OpenCL 1.1 Full profile OpenGL ES 1.1, 2.0 and 3.0
JPEG Enc/Dec
MFC 1080p 60 Enc/Dec
DRAM
LPDDR3 933MHz 32bit 2-port 14,9Gbytes/sec 2GByte PoP
USB 3.0 #0
USB 3.0 #1
eMMC 5.0 8bit
SD3.0 Host
UART #2
I2C #1
UART #0
SPI #1
ADC
GPIO
USB 3.0 DRD/OTG Micro A-B
USB3.0 Host Type A
eMMC Module Socket
MMC 0 8bit
Micro SD Slot
MMC 2 4bit
Serial Console
I/O expansion Port (30pin)
USB 2.0 Host
I2C
I2S
HDMI
eDP
USB Hub + Ethernet Controller
PMIC
CPU/GPU/DRAM Power Monitoring
Audio Codec
Out
In
4 x USB Host Port
Ethernet 10/100
DC 5V/4A
Hedphone / Microphone
HDMI Type-D
DisplayPort

Figure 2.5: Odroid XU3 Block Diagram (source: Hardkernel Website [XU3b]).

of the external interfaces and components available to get more consistent energy readings; the Ethernet port for internet access, eMMC module or MicroSD card for the Operating System (OS) image[5] and file system, and the board energy monitors. The Exynos 5 Octa consists of four big ARM Cortex A-15 and four small ARM Cortex A-7 cores, which share the same ISA but have different characteristics when it comes to performance and power consumption. The Exynos chip also has an ARM Mali-T628 GPU with six cores and combined with the two ARM processor types, which makes the Exynos chip a *three-way heterogenous multicore with 14 cores.*

The Exynos chip makes use of ARM big.LITTLE technology. The technology enables the OS to perform Global-Task-Scheduling; to dynamically assign threads to the most appropriate CPU based on the run-time information [ABL]. The board will compile and run the programs pushed to it by the server script `push.py`, measuring time and energy as the program executes. Upon program termination, the board will calculate the program energy consumption and report results back to the server for further processing. The observant reader might view a compilation both at the server and at the board as redundant. Server compilation is done as the server has the capacity to handle multiple users simultaneously, and provides quicker feedback to the user in case of compilation errors. Also, the system lacks a good cross-compiler, and we have not found an appropriate cross-compiler for the server. The backend therefore needs to compile the submitted program, as the server and backend have different ISAs. The current CMB backend supports the programming languages C and C++, compiled with gcc-4.9 and g++-4.9 respectively. The compilers have support for both OpenCL v1.1, OpenMP 4.0, NEON and PThreads NTPL 2.19.

2.2.4 Energy Measurements

The Hardkernel EnergyMonitor program is used to monitor power consumption and is compatible with Odroid XU3 [OEM]. The pipeline in Figure 2.6 shows the execution pipeline when a program is pushed to the backend. The program is first compiled, and further executed on a small input set to detect potential runtime errors. If there are no runtime errors, the EnergyMonitor program is started and the cache is cleared. Due to energy consumption irregularities found by Follan and Støa [FS15], clearing the cache is necessary to obtain stable energy readings. After the cache is cleared, the CPU is heated using the UNIX `stress` [STR] command. The CPU runs `stress` to obtain a fixed start temperature (currently 60 degrees Celsius) before running the program. Starting benchmarks at the same temperature before each run of a benchmark is important to obtain stable energy readings, as pointed out by Cebrian and Natvig [CN13]. The current backend uses the sensors monitoring the temperature of the four big ARM Cortex A-15 cores to arrive at the given target temperature of 60 degrees. The program is then executed and,

[5]The eMMC module loads the OS image faster than a MicroSD card, and is preferred over a MicroSD card if the module is functional.

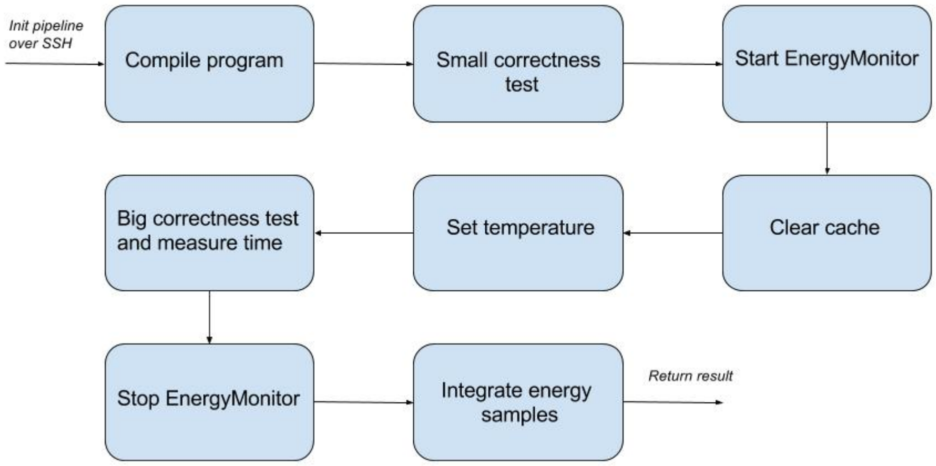


Figure 2.6: Backend execution pipeline (adapted from [FS15]).

upon program termination, the EnergyMonitor program is ended. Finally, after post-processing the program power consumption, the result is reported back to the server.

Timestamps are captured right before and right after a program executes to measure execution time. The same timestamps are also used to fetch the power consumption during program execution, by parsing the output of the EnergyMonitor program and select the measurements captured for the big correctness test in the Figure. Upon receiving the results from the backend, the server then calculates the Energy-Delay Product (EDP) [HIG94]. This metric is chosen as it emphasizes both energy and performance. Energy is in itself a poor energy efficiency metric since we could simply run a program on a slow and under-clocked CPU to obtain low energy consumption. EDP is defined in Equation 2.1, where E is the energy used, D is the delay or execution time.

$$EDP = E * D \tag{2.1}$$

As noted, the Mont Blanc project uses the metric FLOPS/W. This metric may be a useful metric when running the same benchmark or implementation over and over again to test the energy efficiency of HPC and processor architectures. However, CMB compares different implementations on the same problems repeatedly, i.e the number of operations may differ from one implementation to the next. Another problem is to determine the number of *useful* instructions, or in other words, instructions that contribute towards solving the problem. For these reasons, EDP is preferred over FLOPS/W in the CMB system and is also the primary energy efficiency metric used in the system.

2.2.5 Code Correctness and Code Deployment

Unit test frameworks and other tools are used to ensure code correctness in both the frontend and server code. The tools used by the CMB system is summarized in Table 2.1, and are split into four categories. *Package Managers* are used to install and upgrade dependent code packages and libraries used by the system. The required code packages needed are all listed in special files, one unique file per package manager. Both the frontend and server code makes use of *test frameworks* to develop and run unit- and integration tests and there exists multiple unit tests on both system components. *Linters* are special tools that ensure that language standards are followed, such as semicolons at the end of each line in Javascript or correct indentation in Python files. Other tools include `gulp` [GUL], which is a small build system used at the frontend that can execute small user defined tasks, such as running the frontend linter and unit tests or compressing the HTML, CSS, and Javascript files. At the server, `virtualenv` [VIR] is used as a virtual environment as described in Section 2.2.2.

To quickly deploy new features, the system makes use of *Jenkins* [JEN] as Continuous Integration (CI) server or build server. CI is a software engineering practice that automates the testing and deployment of systems. Fowler reports some benefits of CI in his article from 2006; a clone of the production environment (so-called development server) runs tests on the code changes before deployment to production, bugs are discovered and removed easily, and as a whole it makes it possible with rapid integration of new features [Fow06]. As briefly mentioned in Section 2.2.2, the CMB system has a development and production server, and these are used to implement CI as a practice in the project. The development server is used for manually testing new features in a production-like environment as features are developed. Features that passes the manual testing stage are then deployed to the CMB production server. Jenkins can be configured to fit into various system environments, and Figure 2.7 summarizes its use-case in the CMB system.

The Jenkins *build pipeline* in Figure 2.7 is defined for both the frontend and server code. Each step in the pipeline is called a *task*, and may have various configurations and use-cases depending on the system environment. The tasks shown in the figure demonstrates the task configurations used in the CMB system. The first task in the pipeline is launched upon code changes on the master-branch in the respective git-repository. The task pulls all the new code changes from git and ensures that

	Package Manager	**Test Framework**	**Linter**	**Other**
Frontend	npm [NPM] bower [BOW]	Jasmine [JAS] Karma [KAR]	jshint [JSH]	gulp [GUL]
Server	pip [PIP]	nose [NOS]	flake8 [FLAa]	virtualenv [VIR]

Table 2.1: CMB code correctness tools.

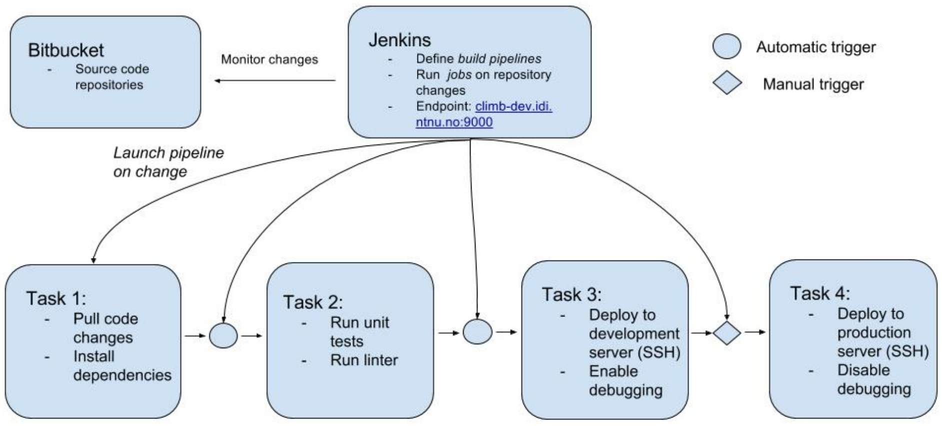


Figure 2.7: Build pipeline.

installation of package dependencies does not crash the system. If the first task completes without errors, it will automatically trigger the next task, which will run all defined unit tests and the linter. As before, if there are no errors, task three is launched, which deploys the recent changes to the development server over SSH with debugging enabled. If the third task is successful, the code changes should be visible on the development server of CMB.

Manual acceptance testing and integration tests can be carried out. If the developer(s) are satisfied with the new features, the final and fourth task can be triggered manually. The task deploys the changes to the production server of CMB, which also takes care of disabling debugging and compressing the frontend code. Disabling debugging and doing frontend code compression reduces the number of requests made by the browser and makes the server code execute faster, and it is important to note that this does not affect the functionality of the system. The development server does, as mentioned, have debugging enabled, and uncompressed code to easier find bugs during acceptance testing. Since the difference between the two servers does not sacrifice the functionality of the system, it does not violate the rules set by CI.

The CMB development server has a Jenkins server installed and provides a user interface available at `http://climb-dev.idi.ntnu.no:9000`. All development should be done locally and then merged with the code on Bitbucket to follow the practice of CI correctly. It is considered bad practice to develop code on the development or production server, and the reader should refer to Appendix E to learn about system setup and local development.

2.2.6 Security

User Information Security A number of security measures are taken to make sure that user data are stored and accessed safely. The password is processed by the Python Werkzeug security package [WER] upon user creation, to ensure that the password is salted and hashed before being stored in the database. This procedure executes upon both regular and admin user creation, and it is therefore very hard for adversaries to reconstruct a given password hash. The user sends the password in the login request made to the server API, and becomes authenticated if the password hash matches the hash stored in the database. The server then returns a *token*[6] that contains enough information to describe uniquely a user-session, and it is sent back in a HTTP response header named *Authorization*. The token is required in some of the requests made to the server API, such as uploading and running submissions, and the frontend code takes care of including the header before such requests are made. The token is valid for one hour, which requires a new login after an hour of inactivity.

Since the token is stored at the frontend, the server is *stateless* and thereby RESTful as mentioned in Section 2.2.2. To make the transfer of information even more secure, Hyper Text Transfer Protocol Secure (HTTPS) is used to encrypt all the information sent in requests between the server and frontend. HTTPS uses Secure Socket Layer (SSL) to encrypt messages sent to and from the server and is almost impossible to decrypt, which enables safe transfer of passwords and other sensitive data between the server and frontend.

Uploaded Programs Security An uploaded program can potentially contain malicious code either in the source files or file names. To restrict the number of possible actions that can be made through the uploaded code, the backend executes the code using a Unix user named *worker* that has minimal OS permissions on the backend. Also, to remove possible malicious scripts residing in file names, the server again makes use of the Python Werkzeug security package [WER] before storing the files in the server file system. It might be worth mentioning that it could be an idea to further extend the system with designated compile servers or threads since unknown errors during compilation might crash the server. Since no such error has occurred in the past or present, it is listed in Appendix C as a possible feature to be added to the system at a later point.

System Security The reverse proxy server used at the server is configured to handle HTTPS requests. As mentioned, the server uses SSL when sending and receiving HTTPS requests. SSL requires an SSL Certificate to work, which is a guarantee that the holder of the certificate provides trustworthy and safe web-content. The most common browsers have a list of trusted third parties issuing certificates, called Certificate Authorities (CAs), and every certificate issued by one of the CAs on the list are trusted as safe. Uninett [UNI] is one example of a CA

[6] A token is some unique string valid for a specific period of time.

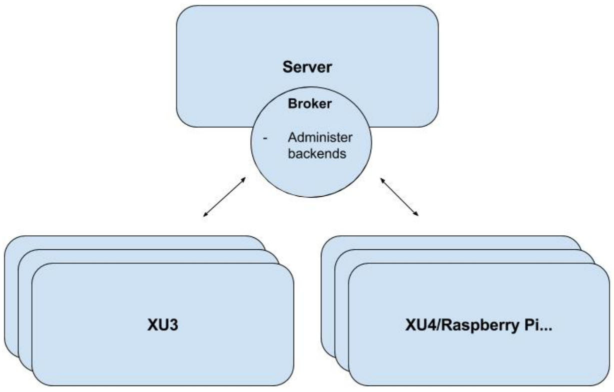


Figure 2.8: Scalable CMB architecture.

and has created the SSL certificate that is used by the CMB system. Since Uninett is a well known CA, the HTTPS requests sent by the system is trusted by most browsers.

The CMB backend also has Uncomplicated FireWall (UFW) [UFW] installed. UFW restricts access to the board by requiring every SSH request to originate from within the NTNU network. The backend also has Fail2ban [FAI] installed, which restricts the number of authentication attempts made within a certain timeframe. If there are more than three attempts from the same IP-address within the configured timeframe, the IP-address is banned from sending requests for 10 minutes. The server also has UFW and Fail2ban installed, but the setup of UFW is slightly different. The server UFW setup allows all requests made to port 80 and 443 to enable the requests made to HTTP and HTTPS respectively. The setup makes the CMB frontend accessible from outside the NTNU network. To keep up to date with security updates `unattended-updates` [UNA] is enabled on the production server, which installs security updates if needed. The check for security updates are made every night at 02:00, and an automatic restart of the system is done if the update requires it.

2.2.7 Related CMB Project - System Scalability

Previous user testing has shown that the system is bottlenecked by the number of executing units. The bottleneck was noticed during the Autumn of 2015 when students had to use the system as part of mandatory exercises in the course TDT4200 Parallel Computing [TDTb]. The specialization project report delivered in December 2015 presented feedback given by the TDT4200 students, and some students pointed out that submissions often stalled for several minutes in the run queue with little feedback on progress. Even though students were advised to start exercises early and submit to the system long before exercises were due, the system should still handle a higher load of users during the last few hours before the exercises' deadline. Further, if the system is to be used in programming competitions like International Collegiate Programming Contest (ICPC) [ICP] or IDIOpen [IDIb], it needs to handle multiple concurrent submissions without stalling them in the queue for too long.

The observations motivated the extension with several execution backends like depicted in Figure 2.8. Christian Chavez has been assigned the thesis on CMB scalability, which should extend the system with more Odroid XU3 boards. The idea is to have multiple backends at the CMB production server to handle multiple submissions at the same time, as described in the CMB paper by Natvig et al. [NFS$^+$15]. A control mechanism needs to be implemented, to administer the incoming requests and possible run queue race conditions. This mechanism is called a *broker* in Figure 2.8. Keep in mind that the figure describes the concept of scaling the system with multiple boards, and there are several possible ways to implement the suggested broker.

Follan and Støa suggested that one could simply add more push scripts, one script for each connected backend [FS15]. While this solves the problem of extending the system with more boards, the push script has proven to fail and terminate due to some erroneous output from the backend. Finding errors and bugs in bash scripts executed over SSH in Python has proven to be difficult to debug, which has driven the project team to look for other solutions than simply extending with more push scripts. The Specialization project report proposed the use of *task queues*, which are mechanisms to distribute work among threads or machines. However, it might impose unnecessary complexity to the system.

The thesis will also describe how the broker can be used to allow different backends other than the currently used Odroid XU3 board. Figure 2.8 also shows an illustration of the architecture if extended with different backends. Allowing various types of boards on the system enables further compelling research on energy efficient programming, and may provide CMB users with the possibility of comparing the energy efficiency of benchmarks on multiple architectures in a quick manner.

2.3 Related Work

This section will present related OJs which might inspire the future development of the CMB system. The presented work is inspired by the OJs presented in the master thesis of Follan and Støa and further extended with interesting OJs as well as the OJs mentioned in the specialization project. There exists many OJs that have a vast amount of users and submissions, but as mentioned, we are not aware of any other OJ focusing on energy efficient programming. Instead, we present popular OJs and their features, which might have an effect on the future development

Definitions	
Online Judge	"An online based computer system, providing programming problem descriptions and data sets to automatically judge wether a particular solution solves a given problem." Inspired by definition found in [KLC01]
Crowdsourcing	"The act of taking a job traditionally performed by a designated agent and outsourcing it to an undefined, generally large group of people in the form of an open call." [CRO].

Table 2.2: Defining OJs and Crowdsourcing.

of CMB. Further, we also present the most popular crowdsourcing sites as these have some relevance to the project. Table 2.2 lists the definitions of an OJ (briefly mentioned in Section 2.2) and crowdsourcing. Since crowdsourcing is a broad term we will further restrict the definition: *Crowdsourcing is the action of outsourcing programming tasks and problems to a large group of programmers in the form of an open call.* The definitions are valid throughout the thesis.

2.3.1 Online Judge Systems

Kattis

The first version of Kattis was developed in 2005 at KTH in Stockholm [EKN$^+$11]. The judge was first used for assessing programming exercises in various courses at the university. Since the first version, the Judge has developed into a sophisticated and well known OJ and is widely used in the Nordic countries, perhaps because of its usage in the programming contests ICPC [ICP] and IDIOpen [IDIb]. The judge supports 13 programming languages and offers a broad range of programming problems to solve with varying difficulty. The judge is offered to the public through a website, but it is also offered as a simple Command Line Interface (CLI).[7]

Kattis also offers services for firms and universities alongside hosting a public OJ. Professors can register courses and automatically grade programming exercises, such that students easily can get feedback and track their exercise scores. The service also offers plagiarism checks of submitted code and analytics of the registered courses. Companies can register to challenge potential job candidates with a set of programming problems before interviewing them, and offer crosschecking of submitted code to filter out cheaters. The universities can choose either a free subscription with a limited number of registered courses and teachers, or a premium subscription with an unlimited number of teachers and courses with a cost of 15$ per student, per course. Companies have three different paid subscriptions

[7]The Kattis CLI can be found at github: `https://github.com/Kattis/kattis-cli`.

to choose from, which offer either a few, medium or large number of interviewers and problems depending on the chosen subscription.

UVa Online Judge
The first version of UVa Online Judge [UVA] was developed by former student Ciriaco García de Celis in 1995 [RML08]. The judge was first built as a series of bash scripts, but the scripts were later replaced by a team of students to make the judge able to handle the increasing amount of submissions and to ready the system to be used in programming competitions. Today the UVa Online Judge is very popular, with over 1,8 million submissions registered in 2015 and over 100,000 users. The judge also offers over 4,300 programming problems solvable in 6 different programming languages.

PKU JudgeOnline
PKU JudgeOnline [PKU] was released in 2003 and is one of the OJs with the highest number of submissions. At its peak in 2010, the judge had over 1.768 million submissions, but the number of submissions has decreased in recent years measuring almost 1.3 million submissions in 2015. The OJ provides mostly the same features as other OJs, like viewing statistics, submitting code to problems, and hosting online contests.

URI Online Judge
The URI Online Judge is served through a website which was first presented in July 2012 [BFT13]. The OJ aims to support both teaching student about programming and professors aiding them to manage courses and exercises. A total of eight problem categories are offered, training students in various topics in programming. Bez et al. mentions a couple of features implemented. One feature lets the user send in the input test cases to a problem, in which the online judge responds with the correct expected output. Another lets the user view the code directly in the browser during compilation errors, with highlighted code lines on those lines containing errors [BFT13].

HackerRank
HackerRank [HACb] has over 800,000 developers registered and over 800 public problems and challenges. The project was started in 2008 by Vivek Ravisankar and Hari Karunanidhi. The founders felt like they spent too much time on engineering interviews and less time creating great solutions, and they also had a hard time finding good programmers through the traditional interview process. The OJ supports over 35 programming languages and offers a clean design in their interface, with features such as online code editing. HackerRank does also have an extensive work section with more than 1,000 companies registered. The companies interested in filling a particular position can create a basic free profile to interview candidates, create or add problems, and watch the candidate code in real-time. One can also request a premium subscription, which has even more advanced features like de-

tecting code plagiarism. It also has a section for schools that want to create online programming assignments similar other judges providing the same feature.

HackerEarth
HackerEarth [HACa] has over 14.4 million submissions, 3,146 practice problems, and 232 in-depth tutorials. The HackerEarth team has one goal: make technical recruitment straightforward and efficient. The judge offers much of the same features as HackerRank and the other OJs mentioned above but has some unique features as well. They provide a RESTful API documentation, which can be used to integrate the OJ with other software systems. The HackerEarth team has also developed a Google Chrome extension to notify their users about upcoming programming competitions and events. The judge also hosts programming competitions, provides practice problems, provides a similar extensive work section as HackerRank, and provide multiple technical and non-technical blogs. HackerEarth can also be seen as a partial crowdsourcing site, with its numerous challenges and hackathons provided by firms on the HackerEarth website.

LeetCode
LeetCode [LEE] has about 52,000 users registered. Their goal is to prepare coders for technical IT interviews, and offers much the same features as the other OJs presented in this section. However, they have a unique feature which is in-depth articles per problem. Each article goes into depth about the theory required to solve the given problem, and the OJ also enable its users to discuss the articles with other users through their online forum.

Other popular OJs
There are four more Online Judges that is worth mentioning briefly. These are Timus Online Judge [TIMb], A2 Online Judge [A2O], CodeChef [COD], and Sphere online judge [SPH]. These OJs offer features similar to those already presented, so they are not explained in great detail. More information about them can be found on their websites.

2.3.2 Crowdsourcing Sites

TopCoder
The crowdsourcing site that seems to be the most popular is TopCoder [TOPb]. The site has more than 895,000 members and is used by companies like Amazon, Facebook, IBM, and Microsoft for crowdsourcing real world problems to solve. The best solutions to a problem are often awarded with a money prize.

RecSys Challenge
The RecSys Challenge [REC] is a crowdsourcing competition that has been hosted every year since 2010. The competition aims to solve different challenges within recommender systems, where the top three winners receive a money prize and an

opportunity to present their solution at the RecSys Conference. Many students at NTNU compete in this competition as a part of their master thesis.

Chapter 3

Climbing Mont Blanc Usability Goals

This chapter starts with a definition of usable software systems in Section 3.1. The section will define usability, and show a few aspects and characteristics in other OJs, which makes them usable with respect to the definition presented. Section 3.2 concludes the chapter with a discussion of the importance of the usability goals set by this thesis.

3.1 Usability in Online Judge Systems

Usability is defined in the ISO 9241 standard Part 11 [ISO98] as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." The definition is broad and covers many aspects of a product, or in our case a software system. Usability is a broad term and can be hard to define precisely, however, literature seems to agree that the following five characteristics describe a usable software system [Hol05, FJWC01]; *learnability*, *efficiency*, *user retention over time*, *error rate*, and *satisfaction.* A quick summary of each characteristic is summarized below.

Learnability: The user's ability to learn to use the system. Learnability also involves the user's ability to gain efficiency in using the system and reach their objectives in a quick manner.

Efficiency: Users should obtain a high level of productivity when using the system. The usability is improved if the users are able to quickly reach their goals when using the system.

User Retention Over Time: The user should be able to return to the system after a break from using it, and remember the core functionality of the system. A usable software system makes it easy to get back into an efficient state without the need to learn the core functionality of the system anew. The characteristic is also referred to as *memorability*.

Error Rate: The number of errors a user makes along the path of actions before reaching his or her goal. A low error rate among users improves the usability of the system.

Satisfaction: The user's subjective thoughts about the system as well as making the system pleasant to use. Satisfaction may involve functionality that is both visible and invisible through the system frontend.

Another related term to system usability is the notion of *affordance* of things, which is described by Donald Norman [Nor88]:

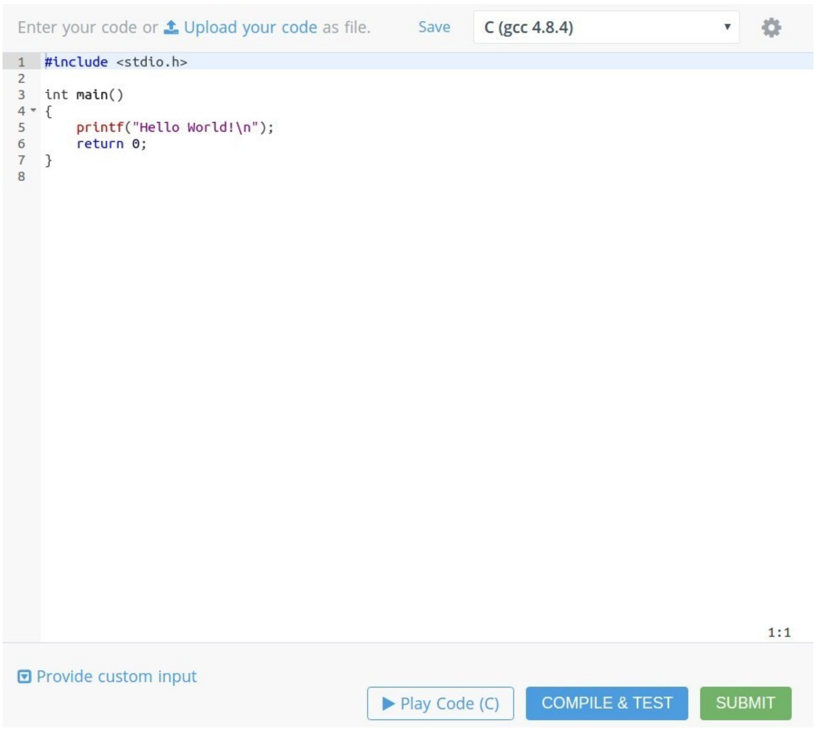
> "...the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could be used [..] Affordances provide strong clues to the operations of things. Plates are for pushing. Knobs are for turning. [..] Complex things may require explanation, but simple things should not. When simple things need pictures, labels, or instructions, the design has failed."

Good affordances of entities and components in software systems will improve learnability, efficiency, memorability, and user satisfaction while lowering error rates, and thereby improve the usability of the system.

The definition of OJs as presented in Section 2.3 is quite straightforward. However, because of its simplicity, it is fairly important that the OJ is highly usable when users are actively using the system to solve programming problems. The most time-consuming tasks done by most users in an OJ, at least in the CMB system, is to read problem descriptions as well as submit code for automatic judgement. The time spent on reading and submitting is minimal compared to the time spent on developing code, and code development often happens off site in an offline environment if not explicitly offered by the OJ[1].

The action of uploading code is one of the main actions done by users. Making the action of uploading code simple is important for both efficiency and learnability. OJs like HackerEarth makes it easy to upload code, providing an online code editor, which can compile and test the code or directly submit a solution like shown in Figure 3.1. The user also has the option of uploading source files directly, which makes it easier for the users wanting to submit source files instead of coding directly in the browser. Presenting multiple alternatives in a structured way to users makes it simpler to use the system, as a given user may choose the method he or she is most comfortable with. The upload feature also shows good affordance, and

[1] Some of the mentioned OJs have web based code editors, such as Kattis [KAT] or HackerEarth [HACa].



```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

Figure 3.1: HackerEarth Code Upload (source: HackerEarth Website [HACa]).

the user instinctively knows how to submit code.

It is also important that the user receives feedback on the state of submitted program. Tracking the state and repeatedly reporting to the user whenever the state of the submission changes is critical. It does not have to be very detailed, as long as the user gets some notion of the state of the program. For example, Kattis does not automatically report the state of a submitted program, but they display a type of progress bar that is updated whenever test cases pass, and the user manually refreshes the current view. However, the more information displayed about critical data such as the state of the submission, the better.

The different components in the user interface should also respond as expected and give logical feedback. Failing to do so might distract the user, and thereby blocks the user from reaching his or her goal. Norman mentions in his book that it is important that all actions made against the system should “give an immediate and obvious effect” in the form of feedback to the users [Nor88]. For OJs, this means clearly stating feedback and making the feedback visible for the users, as well as presenting useful and descriptive feedback messages.

Administrators are also a part of the user group. Usability in OJs also involves making the system usable for system administrators and problem makers, and it is important that admin interface usage is simple and understandable. Usage of a given admin interface likely requires some training and knowledge about the system, but the interface should be simple, so users do not have to relearn how to use the system and quickly achieve high efficiency, i.e., high user retention over time.

Other features implemented by the judge should also behave as expected and the interface should be logically built. The OJs presented in Section 2.3 offer various features, and going into depth about the usability of these features is outside the scope of this thesis. However, it is worth mentioning that the OJs system components should built with good affordances as described above. The professional OJs mentioned in Section 2.3 all have teams of developers and designers and offer simple and usable interfaces for their judges.

3.2 Climbing Mont Blanc Usability Goals

Section 1.2 presented the usability goals for this thesis. The listed usability objectives were chosen as the objectives are important for system usability, as well as some features' potential for further development of the CMB prototype. This section further elaborates on the objectives' importance, anchored in the theory presented in Section 3.1 and to some extent the features' potential for future development of the system.

The objective `U1` concerns the bugs and known issues when starting this thesis in January of 2016, and is divided into three sub-objectives. Fixing bugs and known issues is important and, in some cases, crucial to offer a system with good usability. Making it possible for Mac OS X users to upload code to the system, i.e., solving objective `U1.1`, is crucial for usability in the CMB system. If users are denied uploading to the system even with a seemingly correct zip-file, it is likely that their satisfaction of the system will be low. As the CMB team aims to offer the OJ to as many people as possible, with support for the major operating systems and browsers, it is therefore unacceptable to deny users using Mac OS X access to the system.

Submissions should not be able to block execution on the board and be locked in a running state (`U1.2`). The bug does not only give poor feedback to the user who submitted the problem, but it also blocks other users from using the system. The efficiency and satisfaction of other users trying to use the system would likely be low, as they are not given any feedback on the situation. The last issue concerns showing hidden submissions upon sorting on different highscore list metrics (`U1.3`), and is considered a nice-to-have fix and is less crucial compared to the other issues presented here. However, it does affect the general satisfaction with the system and it may increase the error rate and confusion for some users.

The objective `U2` is concerned with further extending the system with usability features. The five usability sub-objectives aim to further improve or extend the usability of the system, either by adding new features or enhance existing system components. The objective `U2.1` aims to improve the feedback presented to users by making feedback more structured, clear, and visible, especially during error messages. Immediate and obvious feedback of actions is, as mentioned before, important and it may help users gain efficiency and better the users' learnability of the system. Actions that require time to finish executing, such as running submissions, should give some feedback on the state of the action. Preferably, some feedback should also be given to indicate when the action is done executing.

Implementing real-time updates of data is related to giving good feedback and is the goal of objective `U2.2`. The feature makes it possible to automatically update the data displayed in the view, which for most users makes the system more satisfiable to use as they no longer need to manually update the web page to receive up-to-date data. It also displays good affordance, as users may expect the frontend views to change as data changes. The feature also has huge potential and can be used when further extending the system with more features. Some examples include detailed statistics of expected queue time and notifications when a submission's queue position changes. Section 7.2 mentions some possible future extensions, which may use the real-time update technology in their implementation.

Newly added input and output files to problems should behave as expected and submissions to the problem should have a predictable outcome (`U2.3`). The objective is important to both administrators and regular users. Administrators might become unsure of the functionality of the admin interface, and it also lowers satisfaction and increases the error rate on adding problems among system administrators. Users might begin to doubt their own, possibly correct, code depending on system output on the problem, which lowers efficiency and learnability while the error rates increase.

The last two objectives are mainly focused on user satisfaction. Objective `U2.4` aims to develop a bulletin board to display administrator messages. The feature makes it possible for administrators to notify users about important events such as system maintenance etc. or less important events such as newly added problems. Objective `U2.5` focuses on displaying only necessary information to users, which also helps users focus on the information that is important. Furthermore, the objective also aims to improve the overall satisfaction with the user interface.

Chapter 4

Climbing Mont Blanc Improvements

This chapter describes the implementation of the usability and improvement goals presented in Section 1.2, as well as a short discussion of the proposal objectives presented in the same section:

- Fix bugs and known issues in the system, i.e, objective `U1`.
- Change the existing DBMS into a more sophisticated one, i.e, objective `I1`.
- Implement and extend the CMB system's usability features in accordance with the CMB team, i.e, objective `U2`.
- Propose improvements to an existing stability test, improvements to the HowTo-page and existing provided problems, and implementation of a discussion forum, i.e, objectives `P1`, `P2`, and `P3` respectively.

A frontend data model is in the following sections defined as an entity that stores data. A data model also contains control logic to modify or update stored data, and ,therefore, also acts as a *controller*. For example, fetching data from RESTful APIs is common in the CMB prototype frontend data models. Each model is also coupled with a frontend view, which uses the internal logic of the data model to present the model data to users.

4.1 Real Time Updates

The frontend view changes when a user performs actions against the frontend models as mentioned in Section 2.2.1. However, the frontend view presented to a given user does not update automatically as other users interact and change their data models, as models are stored locally in each of the user's browsers. If the updated model contains data that should be known[1] to all users, the users do not get notified about the model changes dynamically and views may, therefore, display out of date information. Figure 4.1 shows an example of the problem, as Alice updates some of her browser's model data when she interacts with the system. If Alice changes some data present in Bob's models, Bob will not be notified of the changes as all data transfers are done with HTTP requests between Alice and the server. However, Bob can fetch up-to-date data by manually refreshing his web page.

Many websites require shared data between multiple clients and dynamically notifying clients if there are changes to the data. As an example in the CMB system, it would be nice if the frontend interface dynamically updated as a user's submission finished running at the backend. In system version one the user had to update the data manually by clicking a refresh button provided by the user interface or manually refresh the web page, to fetch up-to-date model data. However, this section presents a technology which has been introduced in version two of the system to dynamically update data relevant for multiple clients.

Socket.io [SOC] is an API for enabling real-time communication between the server and connected clients. The API was first made as Javascript library, but many open source projects have developed modules for other programming languages integrating the API. One benefit of the API is that it works as a wrapper around a set of real-time communication protocols to enable support for different browsers, which means that the framework can automatically detect the protocol supported by a client and use that information to select the best-suited communication protocol.

Rohit Rai states the communication protocols supported by the Socket.io API [Rai13]. As many protocols are available, Figure 4.2 only shows the communication protocols enabled in version two of CMB. The WebSocket protocol [FM11], shown in Figure 4.2c, has become more popular since its introduction in 2011 and is now supported by the most popular browsers. The protocol is a bit different from the well known HTTP protocol, as there is a persistent connection, or socket, between the client and the server as long as both entities are connected to the socket. The socket connection is closed if the client closes the browser window or the server goes down, or if the code explicitly indicates to close the socket. WebSockets enables easy two-way communication by letting two entities emit messages back and forth on the socket connection and respond differently depending on the type of message emitted on the socket.

[1]Hereby known as *shared data.*

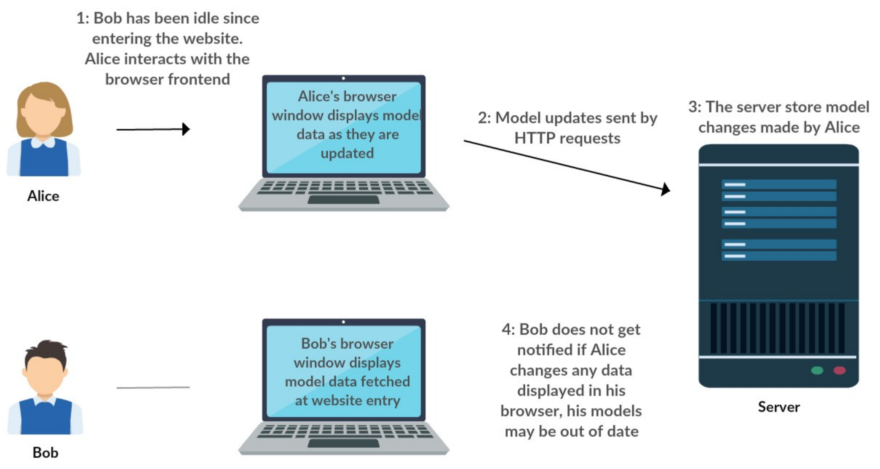
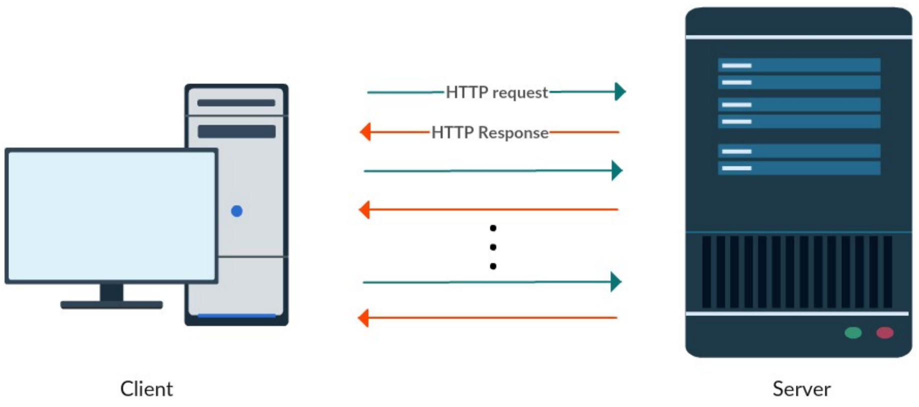

1: Bob has been idle since entering the website. Alice interacts with the browser frontend
Alice
Alice's browser window displays model data as they are updated
2: Model updates sent by HTTP requests
3: The server store model changes made by Alice
Server
Bob
Bob's browser window displays model data fetched at website entry
4: Bob does not get notified if Alice changes any data displayed in his browser, his models may be out of date

Figure 4.1: Example of updating models without model change notifications.

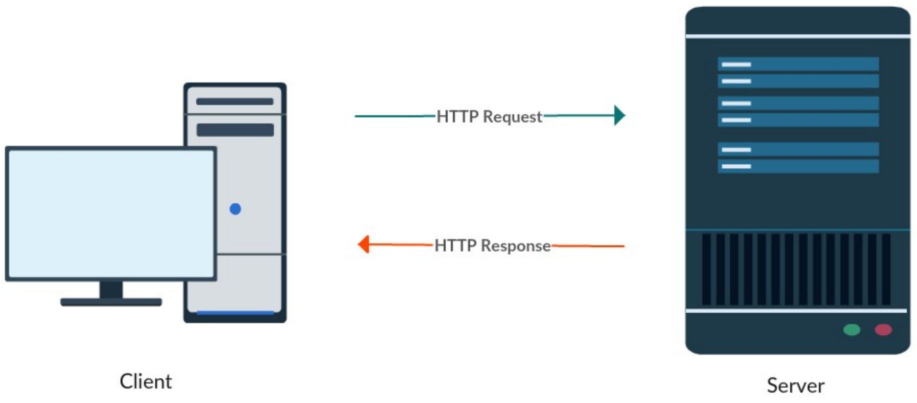
Polling and long-polling both use the HTTP protocol and may seem similar at first glance. However, in long-polling, the server keeps the connection between the two entities open until there is an update to the requested data as shown in Figure 4.2b. In polling, as shown in Figure 4.2a, the client continuously requests data with some constant delay between each request, while the server responds on each request with the data currently stored at the server. The three presented protocols all solve the problem of real-time notifications. However, to make the below discussions more explainable, we can think of the connection between the server and the client as a "socket", which is a persistent communication channel with the equal functionality regardless of the underlying protocol.

Each socket can be "*namespaced*" into separate communication channels within an application. A namespace can be viewed as an endpoint or network path, for example in version two of the CMB prototype the namespace */cmb* is used as default namespace to emit messages between the client and the server. Every client initiating a socket on the namespace receives all messages emitted to the namespace, which makes it easy to share information between the server and clients connected to the namespace.

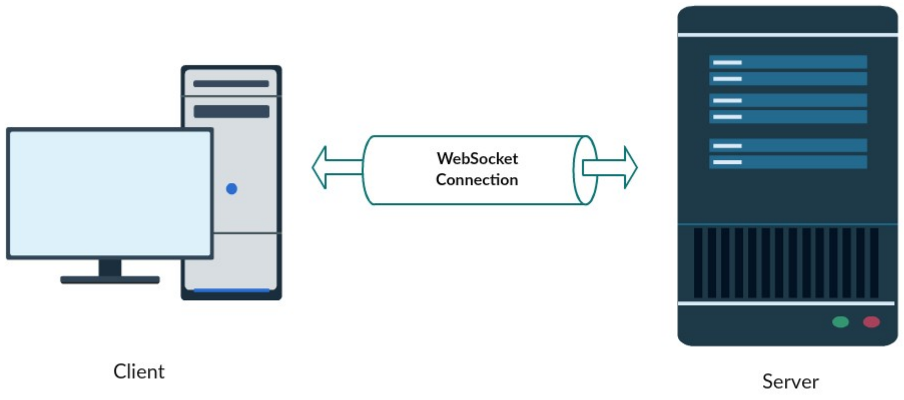
Each namespace can also define a set of *rooms*, which can be viewed as sub-channels of a given namespace. A client needs to join a room explicitly to receive the messages emitted within a sub-channel. Figure 4.3 illustrates the concept. Each message emitted to the namespace */cmb* is received by both client one and two. Client three will only receive those messages emitted to the namespace */test*. If a message is emitted to room one within the */cmb* namespace, only client one will receive the message.



(a) Polling: The client sends a series of HTTP requests and the server responds on each request.



(b) Long-polling: The client sends an HTTP request to fetch new data, the server holds the connection open until the requested data is updated.



(c) WebSocket protocol: a two-way connection tunnel open throughout the whole client session. TCP is used as a transport protocol.

Figure 4.2: CMB Socket.io communication protocols.

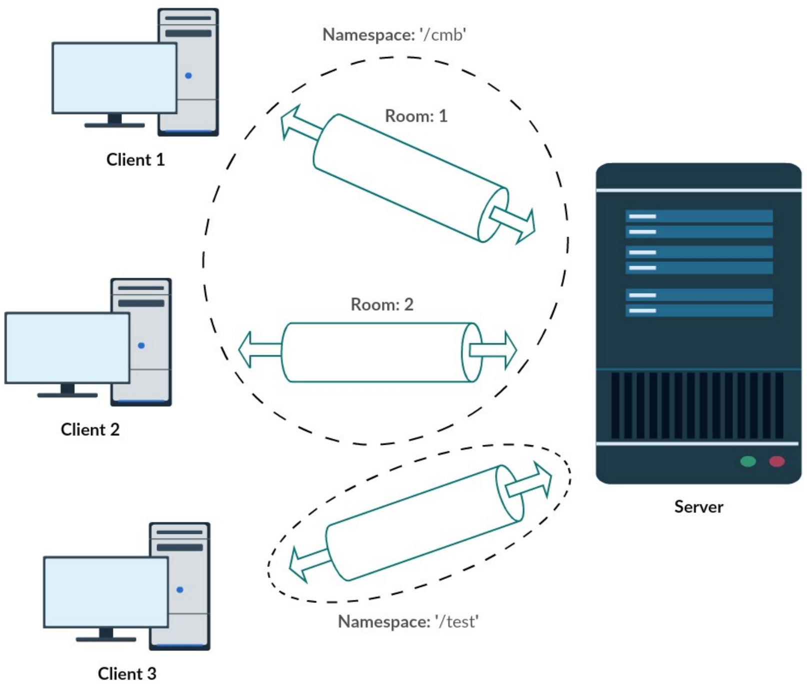
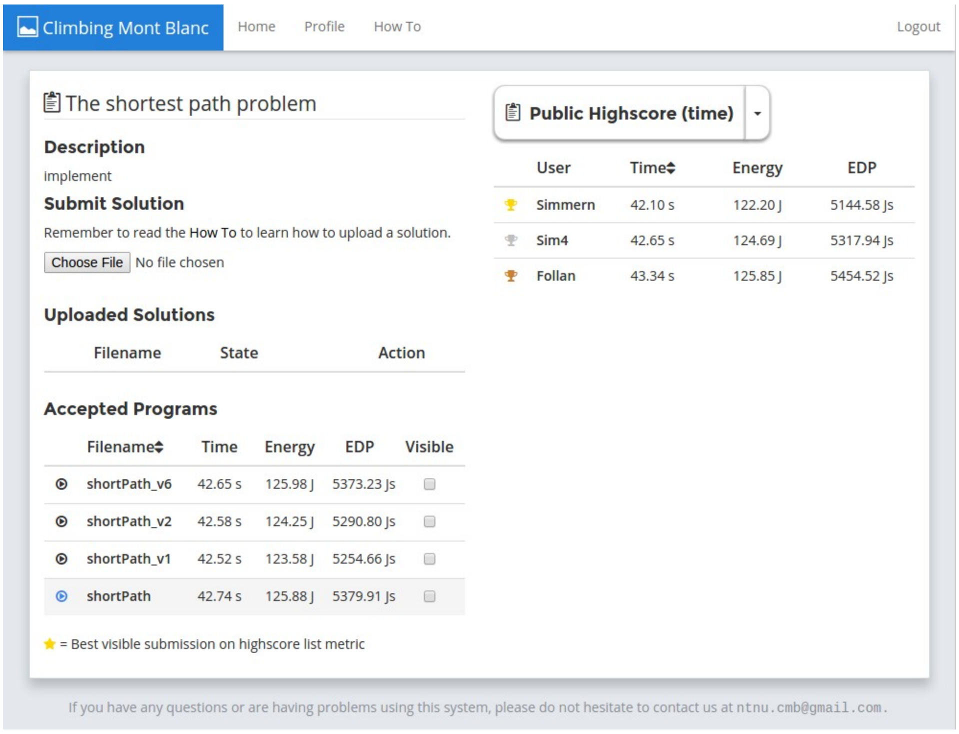


Figure 4.3: Socket.io namespaces and rooms in the CMB system.

Socket.io also lets the programmer define *events*, which can be emitted between the server and connected clients. Table 4.1 shows the events currently defined for the CMB, and the actions taken either by the server or client depending on which entity that emitted the event. The actions are only taken if the code defines event listeners for each event, and for the frontend, it depends on the view that is displayed in each user's browser. As an example, the submission events shown in Table 4.1 are only valid if a given user is viewing the problem-screen shown in Figure 4.4. This makes sense because we are trying to do real-time updates to the highscore-list and the user's submissions while he or she stays in the problem screen. Events will, therefore, not be received if the user navigate to another view, as Socket.io event listeners in the frontend code are bound to views or rather the views' controllers.

The two following sub-sections describe the technologies used by the server and frontend to support real-time communication with Socket.io.



The shortest path problem

Description

implement

Submit Solution

Remember to read the How To to learn how to upload a solution.

Choose File No file chosen

Uploaded Solutions

Filename	State	Action

Accepted Programs

	Filename	Time	Energy	EDP	Visible
	shortPath_v6	42.65 s	125.98 J	5373.23 Js	☐
	shortPath_v2	42.58 s	124.25 J	5290.80 Js	☐
	shortPath_v1	42.52 s	123.58 J	5254.66 Js	☐
	shortPath	42.74 s	125.88 J	5379.91 Js	☐

★ = Best visible submission on highscore list metric

Public Highscore (time)

	User	Time	Energy	EDP
	Simmern	42.10 s	122.20 J	5144.58 Js
	Sim4	42.65 s	124.69 J	5317.94 Js
	Follan	43.34 s	125.85 J	5454.52 Js

If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

Figure 4.4: The updated problem-view.

Event Name	**Emitted by**	**Received by**	**Action**
join-problem	Client(s)	Server	Server adds the client to the problem room specified in the emitted message.
leave-problem	Client(s)	Server	Server removes the client from the problem room specified in the emitted message.
submission-deleted	Server	Client(s)	Client fetches updated submission data from the server.
submission-enqueued	Server	Client(s)	Client displays a message in the browser window, notifying about queue size and information present in the emitted message.
submission-dequeued	Server	Client(s)	Client displays a message in the browser window, notifying about queue size and information present in the emitted message.
submission-finished	Server	Client(s)	Client loads own submissions as well as possible updates to the highscore list.

Table 4.1: CMB Socket.io events and corresponding actions. Each submission event is sent to a room, which simply is the database id of a given problem. Since we have one room per problem, we know that all submission events within a room have information about submissions made to a given problem. Remember that client views need to define event listeners to take action upon events, and in our case, the submission events above are only defined in the problem view.

4.1.1 Frontend Technology

The frontend uses the client Socket.io library [SOC] through an existing Angular component called `angular-socket-io` provided by Brian Ford [For13]. The component is used to define an Angular *factory*, which is simply a component that gathers and structures the setup of Socket.io in the client into a single service. The defined factory can then be used by other components in the Angular app, by marking components as dependencies of the Socket.io factory.

4.1.2 Server Technology

The Python module Flask-SocketIO [FLAf] enables Flask applications to use the Socket.io API. Also, the modules gevent [GEVa] and gevent-websocket [GEVb] is installed in combination with the Socket.io module to enable the use of the WebSocket protocol. Gevent is a coroutine-based networking library providing a high-level synchronous API on top of an asynchronous webserver. Asynchronous coroutines are used to handle multiple concurrent requests from multiple clients and are required by the Flask-SocketIO module to support the WebSocket transport.

As mentioned in Section 2.2.2, Gunicorn uses a number of *workers* to handle multiple concurrent users. To fully support the WebSocket protocol, Gunicorn needs to make use of a custom gevent worker instead, which supports the WebSocket protocol. As stated by Miguel Grinberg on the Flask-Socketio documentation website, Gunicorn can only enable one worker due to limitations in the load balancing algorithm implemented by Gunicorn. The Gunicorn load balancing algorithm cannot simply handle multiple gevent workers and persistent WebSockets connections, which limits us to one worker per server. However, the worker uses coroutines to handle multiple concurrent requests as stated above. The gevent worker is enabled on the development server of CMB, which also required some changes to the Nginx setup to entirely support the WebSocket protocol. Appendix E describes setup information necessary to setup servers with Gunicorn and Nginx, while Section 6.1 discusses pros and cons with selected technologies as well as presenting alternatives to the selected technology stack.

4.2 Frontend

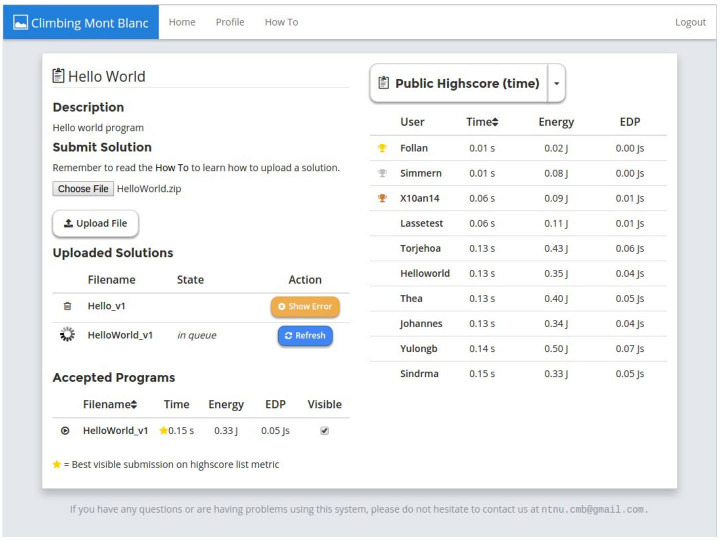
This section presents figures of the views in the frontend that have changed in version two of the system. Appendix B shows screenshots of versions one and two of the system frontend for comparison. The following sub-sections will explain the improvements and updates done to some of the views. Other views in the system not mentioned in the following sections are not discussed in detail as their functionality nor appearance has changed drastically but have rather been changed to keep the views' styling consistent throughout the system.

4.2.1 Bug Fixes

Feedback from previous user testing showed that Mac OS X users were unable to upload files to the system. The system requires the user to compress a folder containing all the source files, which has confused some users. However on OS X, users who seemingly constructed a correct zip still were unable to submit files to the system. It turned out that the most regular way of creating zip files on OS X also sometimes generated a hidden directory called *__MACOSX* and a hidden file called *.DS_Store* within the zip-file. This made the upload fail, as the server assumes it receives a single compressed folder, and not multiple directories within the uploaded zip. Users were also presented with an error message which contained little information about what went wrong.

The upload bug is removed in system version two. Before the zip file is sent to the server, the frontend ensures that the zip file has the required format as described above. The scan also removes all files that are not source or header files and ensures that the zip only contains a single directory containing the source files. The directory *__MACOSX* is also automatically removed if present in the zip. If a user tries to upload zip files with a different format than the format described above, they are presented with an error message telling them to learn more about how to upload their code on the HowTo-page.

The problem-view shown in Figure 4.5 also contained a bug in the highscore list. As can be seen in the figure, users have the option of making their submissions visible in highscore list by ticking the radiobutton in the "Accepted Programs"-table. When sorting on different metrics in system version one, however, submissions marked



Hello World

Description

Hello world program

Submit Solution

Remember to read the How To to learn how to upload a solution.

HelloWorld.zip

Uploaded Solutions

	Filename	State	Action
	Hello_v1		Show Error
	HelloWorld_v1	*in queue*	Refresh

Accepted Programs

	Filename	Time	Energy	EDP	Visible
	HelloWorld_v1	★0.15 s	0.33 J	0.05 Js	✓

★ = Best visible submission on highscore list metric

	User	Time	Energy	EDP
🏆	Follan	0.01 s	0.02 J	0.00 Js
🏆	Simmern	0.01 s	0.08 J	0.00 Js
🏆	X10an14	0.06 s	0.09 J	0.01 Js
	Lassetest	0.06 s	0.11 J	0.01 Js
	Torjehoa	0.13 s	0.43 J	0.06 Js
	Helloworld	0.13 s	0.35 J	0.04 Js
	Thea	0.13 s	0.40 J	0.05 Js
	Johannes	0.13 s	0.34 J	0.04 Js
	Yulongb	0.14 s	0.50 J	0.07 Js
	Sindrma	0.15 s	0.33 J	0.05 Js

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

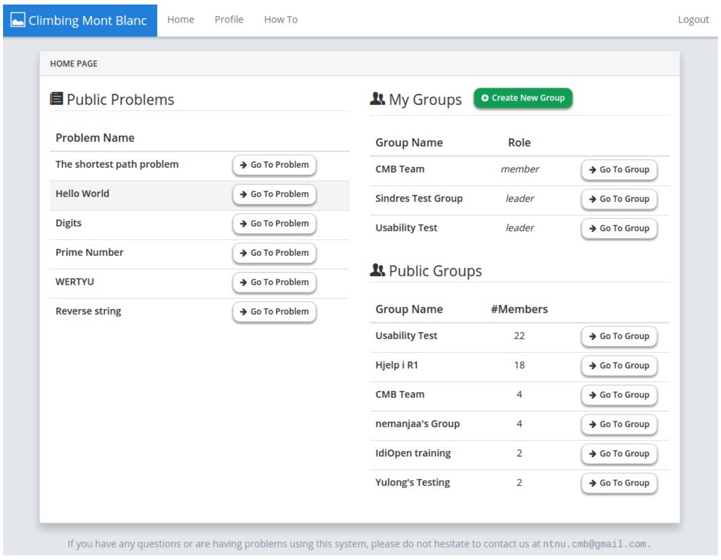
Figure 4.5: The updated problem view.

as hidden in the fetched submissions would suddenly appear in the highscore list. The bug is fixed in the system version two and ensures that hidden submissions are filtered out when the highscorelist is sorted on the different metrics used by the system.

4.2.2 Views and Feedback

The views in system version two have an updated styling. The home-page view shown in Figure 4.6 has a simpler table design and has removed unnecessary information, such as database ids, present in system version one. All screens containing tables have been updated according to the new table design. The problem-view shown in Figure 4.5 has changed drastically compared to version one. In the problem-view of system version one, users had to scroll to the bottom of the view to check out the highscore list. In the updated view, the problem description and file upload functionality are vertically aligned with the highscore list, which makes it easy for users to locate the highscore list when entering the view. Version one of the view also had a single table containing all uploads made by a user, while the updated view splits the old upload table into a “Uploaded Solutions” and “Accepted Submissions” table. The table “Accepted Submissions” contains all submissions marked as finished and successful, while the rest of the submissions are shown in the “Uploaded Solutions” table. Having two tables makes the upload information more structured, and the user can quickly find information about submissions.

System version one displayed information and error messages in the lower right corner of the screen. The messages were very small, and can sometimes be hard to



Public Problems

Problem Name	
The shortest path problem	Go To Problem
Hello World	Go To Problem
Digits	Go To Problem
Prime Number	Go To Problem
WERTYU	Go To Problem
Reverse string	Go To Problem

My Groups

Group Name	Role	
CMB Team	*member*	Go To Group
Sindres Test Group	*leader*	Go To Group
Usability Test	*leader*	Go To Group

Public Groups

Group Name	#Members	
Usability Test	22	Go To Group
Hjelp i R1	18	Go To Group
CMB Team	4	Go To Group
nemanjaa's Group	4	Go To Group
IdiOpen training	2	Go To Group
Yulong's Testing	2	Go To Group

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure 4.6: The updated home page view.

spot as they were only visible for short period of time. Since all messages also have the same styling, it was hard to differentiate between message types. An example of the message component used in system version one is shown in Figure 4.7. System version two displays messages right below the navigation bar as shown in Figure 4.8a, which makes feedback messages easy to spot. The messages are also split into the four message types *info*, *success*, *danger*, *error*, and each message type has their own color as shown by Figure 4.8b. Error messages have also been updated with more descriptive messages if needed.

Most buttons in the frontend of system version two also have symbols inside the buttons. Symbols were added to more clearly state the purpose of the button and thereby improve usability. As an example, the problem-view includes a "play"-symbol inside the "Run Program"-button upon successfully uploading a zip-file. The same "play"-symbol is included in the "Accepted Submission" table to rerun submissions, which improves *learnability* and *memorability*.

System version one required users to go through a two-stage sequence to compile and run a program. The procedure has been further simplified into a single action

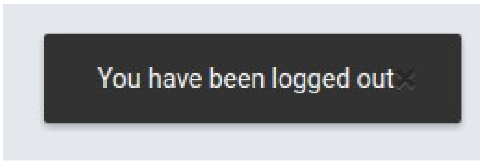
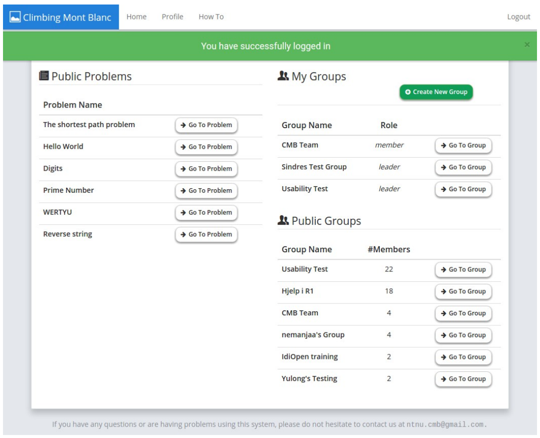

You have been logged out

Figure 4.7: Example of an old popup feedback message.



You have successfully logged in

Public Problems

Problem Name	
The shortest path problem	Go To Problem
Hello World	Go To Problem
Digits	Go To Problem
Prime Number	Go To Problem
WERTYU	Go To Problem
Reverse string	Go To Problem

My Groups

Create New Group

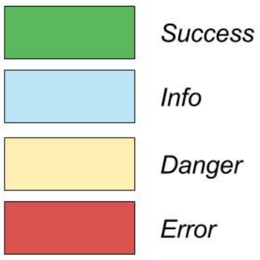
Group Name	Role	
CMB Team	*member*	Go To Group
Sindres Test Group	*leader*	Go To Group
Usability Test	*leader*	Go To Group

Public Groups

Group Name	#Members	
Usability Test	22	Go To Group
Hjelp i R1	18	Go To Group
CMB Team	4	Go To Group
nemanjaa's Group	4	Go To Group
IdiOpen training	2	Go To Group
Yulong's Testing	2	Go To Group

If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

(a) Example of view displaying updated feedback message.



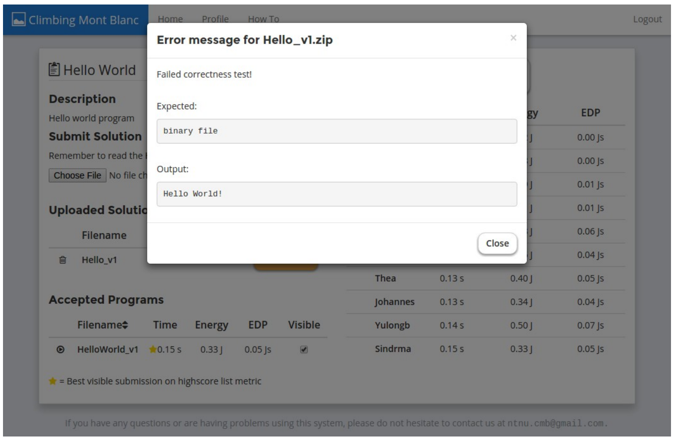
(b) The color of each of the defined message types.

Figure 4.8: CMB feedback messages.

in frontend version two, which makes the action of running a program simpler and more logical. Other buttons in the new version of the interface have symbols as well, for example, a navigation button from the homepage to a problem view includes a right arrow to indicate movement *to* a certain view. Users can thereby infer that a backward arrow means navigating back to the previous view if such a button is displayed in the system.

The updated problem view also displays spinners on submissions currently running or in queue. An example of an active spinner on a submission in the "Uploaded Solutions"-table is shown in Figure 4.5. The table also includes the state of the submissions, which is updated in real-time as the program executes. The real-time updates of a submission's state is possible due to the improvement explained in Section 4.1. If an error occurs during program execution, it is possible to view the error in a popup window, or more technically a *modal*, as shown in Figure 4.9. The content of the modal depends on the type of error and Table 4.2 shows the different error types that can occur during a program's execution. The content of the modal are decided by the "detailed_state" database field in the Submissions table, which is explained in Section 4.3. System version one had an own view displaying error messages, which navigated the user away from the problem view. However, in version two of the frontend, users stay within the problem view removing an unnecessary navigation stage.

A bulletin board displaying administrator generated messages has also been added to the frontend. Figure 4.10 shows an example of how active bulletins looks like in the system frontend, with both removable and non-removable bulletins. Section 4.3 explains the server extensions made to the server code to support the bulletin board feature.



Error message for Hello_v1.zip

Failed correctness test!

Expected:

```
binary file
```

Output:

```
Hello World!
```

Hello World

Description

Hello world program

Submit Solution

Remember to read the

Uploaded Solutio

	Filename
	Hello_v1

Accepted Programs

	Filename	Time	Energy	EDP	Visible
	HelloWorld_v1	★0.15 s	0.33 J	0.05 Js	✓

★ = Best visible submission on highscore list metric

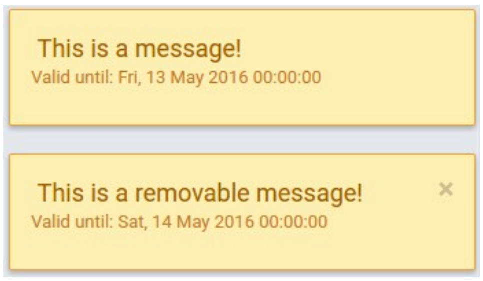
		gy	EDP
			0.00 Js
			0.00 Js
			0.01 Js
			0.01 Js
			0.06 Js
			0.04 Js
Thea	0.13 s	0.40 J	0.05 Js
Johannes	0.13 s	0.34 J	0.04 Js
Yulongb	0.14 s	0.50 J	0.07 Js
Sindrma	0.15 s	0.33 J	0.05 Js

If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

Figure 4.9: Error modal example.

ERROR TYPES	
CORRECTNESS_TIMEOUT	A timeout occurred during small correctness test.
CORRECTNESS_RUNTIME_ERROR	A runtime error occurred during the small correctness test.
CORRECTNESS_BAD_OUTPUT	Output from the small correctness test was not accepted by checker.
PROGRAM_TIMEOUT	A timeout occurred during big correctness test.
PROGRAM_RUNTIME_ERROR	A runtime error occurred during the big correctness test.
PROGRAM_BAD_OUTPUT	Output from the big correctness test was not accepted by checker.

Table 4.2: CMB error types.


This is a message!
Valid until: Fri, 13 May 2016 00:00:00

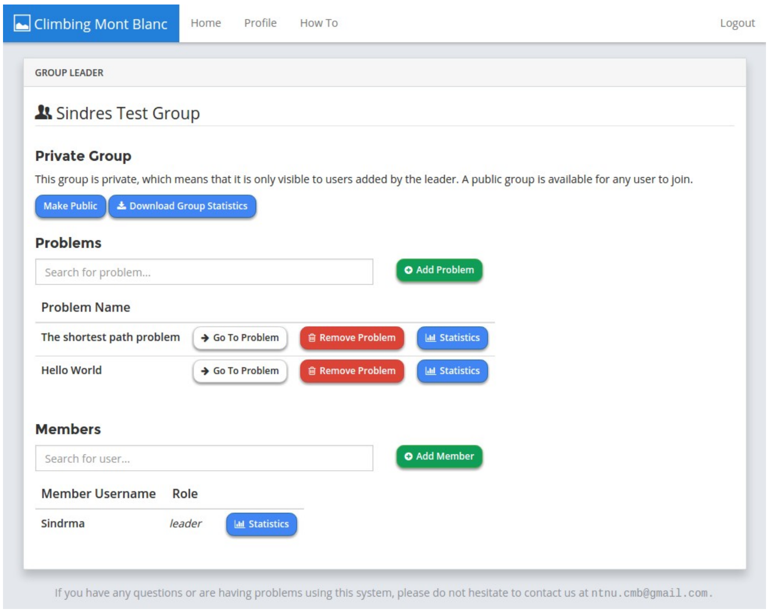
This is a removable message! ×
Valid until: Sat, 14 May 2016 00:00:00

Figure 4.10: Bulletin board frontend examples.

4.2.3 Group Functionality Improvements

Figure 4.11 shows the updated group leader view. The view is displayed if a user navigates to a given group from the homepage and is marked as the leader of the group in the database, as in system version one. Among the visual improvements mentioned in Section 4.2.2, the view has an extra button. The button lets the leader of a group download a JSON file, which contains information about group members as well as submissions made to problems registered on the group. The file can be used to generate group statistics and the functionality was added due to a master thesis of two students, assessing the CMB system's suitability for conducting digital exams in the course TDT4102 Procedural and Object-Oriented Programming [TDTa]. The feature was needed by the students to be used as part of their digital exam experiment with the CMB system.

The updated homepage-view has been extended with a "Create New Group"-button. The button has been moved from the profile view present in system version one into the updated homepage view in system version two, as users may not associate adding groups with profile information. The update also gathers group



GROUP LEADER

Sindres Test Group

Private Group

This group is private, which means that it is only visible to users added by the leader. A public group is available for any user to join.

Problems

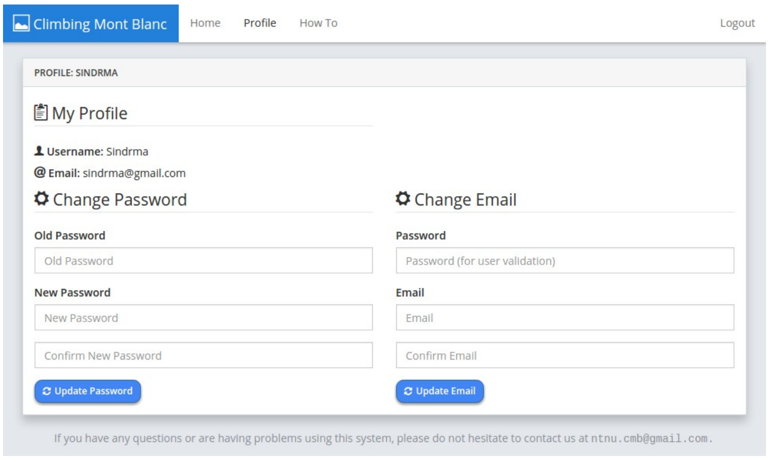
Problem Name			
The shortest path problem	Go To Problem	Remove Problem	Statistics
Hello World	Go To Problem	Remove Problem	Statistics

Members

Member Username	Role	
Sindrma	*leader*	Statistics

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure 4.11: The updated group leader view.



PROFILE: SINDRMA

My Profile

Username: Sindrma

Email: sindrma@gmail.com

Change Password

Old Password: Old Password

New Password: New Password

Confirm New Password

Change Email

Password: Password (for user validation)

Email: Email

Confirm Email

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure 4.12: The updated profile view.

information and functionality, which makes it easier to locate components related to groups. The updated profile view is shown in Figure 4.12.

4.3 Server

4.3.1 Database Management System Updates

Version one of CMB used SQLite [SQLb] as DBMS at both the development and production server. SQLite is a lightweight DBMS, which requires minimal configuration before use and is popular to use during automatic unit testing, which also is done in the CMB system. The initial goal was to improve, and thereby possibly change, the DBMS into a more sophisticated system. However, during the Spring the CMB team found it unnecessary to change the DBMS for performance reasons and we found it more important to extend the system with new features.

However, unrelated to the performance reasons, a new DBMS was wanted by the CMB team and the IDI Department[2] [IDIa]. Therefore, we changed the DBMS to MySQL due to its usage in other applications developed at the IDI Department. IDI provided the databases used by the system, and we also gained access to a database administrator interface available at `https://phpmyadmin.idi.ntnu.no/`. SQLAlchemy made the switch onto the new DBMS easy as it has a predefined MySQL adapter and had no problems in setting up the new databases for our development and production servers. Further, Flask-Migrate [FLAe], used for database *migrations*[3], also worked without any changes in configuration.

A complete database dump was made from the SQLite databases. All SQL `INSERT` statements in the database dump were extracted and executed against the new development and production databases, to move all previously stored data in the SQLite databases over to the MySQL databases. The actions were performed with a combination of terminal commands and the database admin interface provided by the IDI department. As the IDI department have not experienced any performance issues with MySQL and SQLAlchemy, the system uses the standard MySQL adapter provided by SQLAlchemy. Implementing and enabling non-blocking database access has been added back to the backlog[4] found in Appendix C as a performance improvement.

4.3.2 Database Schema Updates

The updates to the previous database schema are shown in Figure 4.13. The Submissions-table has been updated with two new fields. The "detailed_state" field was added to provide more information about a run to the user. In the current improved system, it contains a detailed string about a submission's state as it returns from the backend as explained in Section 4.4. The field can be modified in

[2]The IDI Department cooperates with the CMB team, as they are maintaining the code of the system.

[3]Database migration is the task of managing versions of a database schema without altering previously stored database content.

[4]A system backlog is here meant by a list of features, which should be implemented in the future.

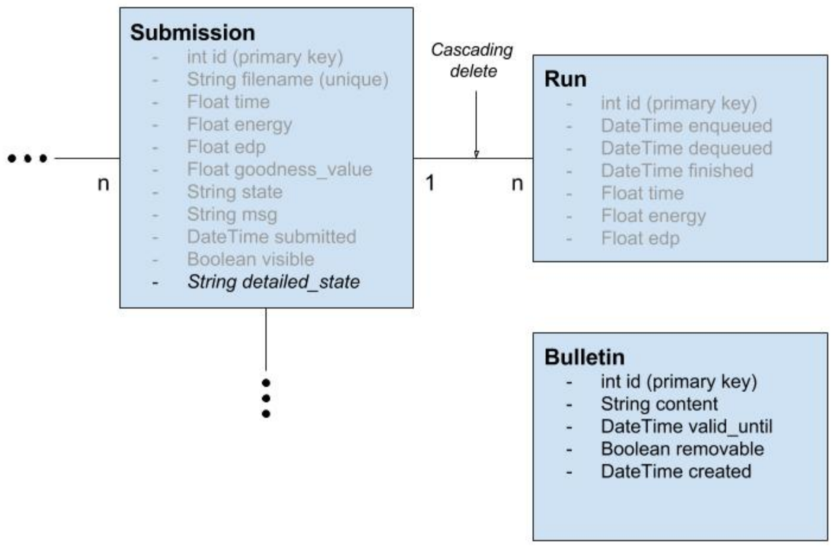


Figure 4.13: Database schema updates. The figure does only include the tables that are updated. Added table columns are highlighted.

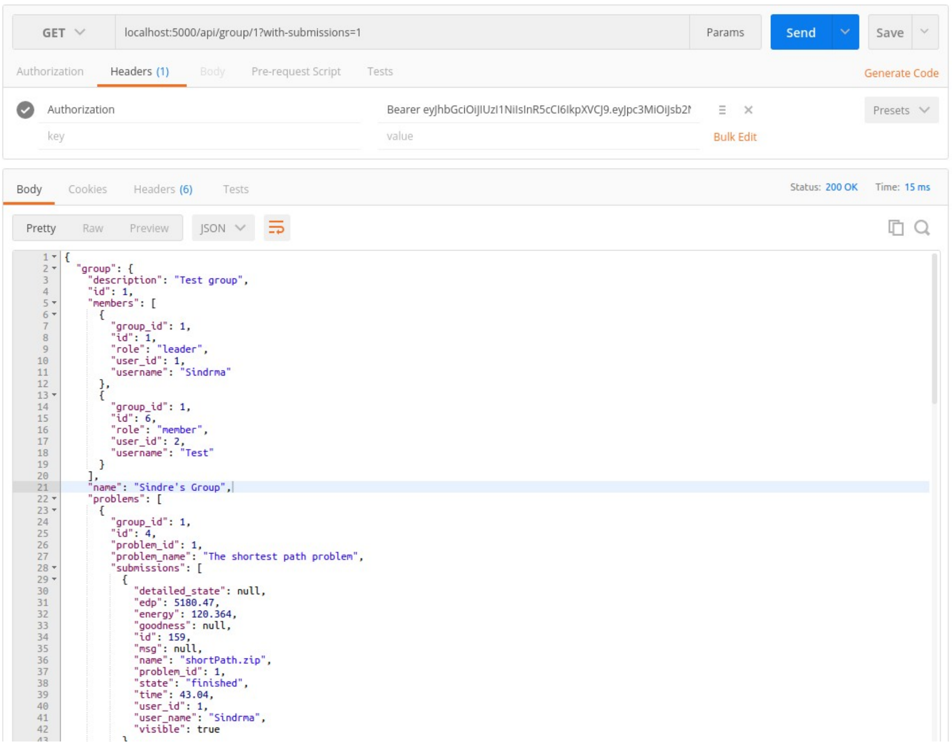
the future to store JSON instead of simple text if more information about a submission becomes available when executing code on the backend. *Future developers should strive against always keeping users up-to-date with the state of their submissions*, and the new field aims to enable such feedback as explained in Section 4.2 above.

A Bulletin-table was added to enable administrators to notify users about news and system messages. Bulletins have a field called “valid_until”, which indicates until which date a given bulletin is valid, and the field is used to select the valid bulletins as explained in the following section and is also displayed in the frontend as explained in Section 4.2 above. Another field called “removable” indicates if a bulletin should be removable or not, and if the bulletin is marked as removable the user is allowed to remove the bulletin from the frontend view.

Cascading delete has also been added between the Submission- and Run-tables. Cascading delete ensures that all rows in the Run table associated with a Submission row gets deleted upon deletion of a submission, which ensures that there are no dangling rows in the Run table, and the database is left in a consistent state after deleting submissions. Database consistency is of importance for both administrators and users, as both user groups can delete submissions.

4.3.3 Endpoint Updates

The API endpoints that have been updated or added are listed in Table 4.3. Figure 4.14 shows an example request against one of the routes defined in the table, which fetches data from a local server. The table does not include changes made to API routes due to the implementation of the real-time update improvement explained in Section 4.1, as the improvements do not change the internal logic of the original routes and has only added code to emit events.



GET localhost:5000/api/group/1?with-submissions=1

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJsb2

Status: 200 OK Time: 15 ms

```
{
  "group": {
    "description": "Test group",
    "id": 1,
    "members": [
      {
        "group_id": 1,
        "id": 1,
        "role": "leader",
        "user_id": 1,
        "username": "Sindrma"
      },
      {
        "group_id": 1,
        "id": 6,
        "role": "member",
        "user_id": 2,
        "username": "Test"
      }
    ],
    "name": "Sindre's Group",
    "problems": [
      {
        "group_id": 1,
        "id": 4,
        "problem_id": 1,
        "problem_name": "The shortest path problem",
        "submissions": [
          {
            "detailed_state": null,
            "edp": 5180.47,
            "energy": 120.364,
            "goodness": null,
            "id": 159,
            "msg": null,
            "name": "shortPath.zip",
            "problem_id": 1,
            "state": "finished",
            "time": 43.04,
            "user_id": 1,
            "user_name": "Sindrma",
            "visible": true
```

Figure 4.14: Example request against local API using Postman [POS]. The request executes the GET-request `http://localhost:5000/api/group/1?with-submissions=1` presented in Table 4.3 with the query parameter `with-submissions` equal to 1 to also include the submissions made by the members of the group.

API Route	**Method**	**Description**
/api/submission/<int:sub_id>	DELETE	If the submission with the given `id` exists and has been created by the currently logged-in user, the submission is deleted from the database as well as the file system. Requires login.
/api/group/<int:group_id>	GET	Fetches submission information about a group by `id` depending on login status. If the user is not logged in, only public group information can be fetched. However, if the user is logged in, the user can also request information about private groups. If the query parameter `with-submissions` is set equal to 1, submissions made from group members are included in the response if the user is either a group member or leader. The response for leaders also include those submissions set to be unvisible.
/api/bulletins	GET	Returns all created bulletins.
/api/bulletins/active	GET	Returns all bulletins currently active.

Table 4.3: CMB API route updates with responses. The routes listed show the substring necessary to query the API, and need to be combined with the *path* of the server providing the API. For example to delete a submission with `id` equal to 1 from the development server API, a delete request needs to be made against `http://climb-dev.idi.ntnu.no/api/submission/1`.

4.3.4 Admin Interface

Several new problems have been added to the system during the Spring semester by various administrators. Most problems were added by the two master students assessing the CMB system's suitability for conducting digital exams. However, during the Spring, we had a lot of trouble making the newly added problems solvable, as some submissions locked the backend infinitely. It turned out that some of the administrators had uploaded input and answer files with CR/LF (DOS) line endings, which locked the checker program during parsing of the input and answer files. As the server and backend is Unix based and uses Unix line endings (LF), the compiled checker program did not handle DOS file types and such files sometimes caused the checker to pause execution during correctness checking.

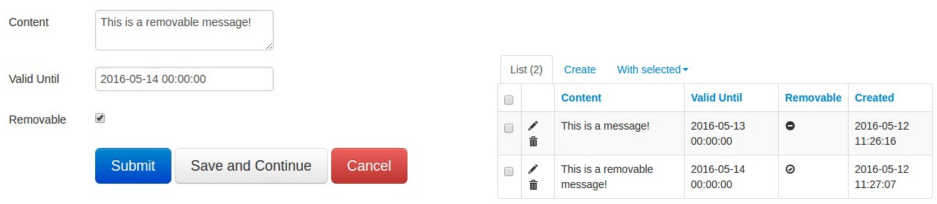
The bug is removed in the updated admin interface. When uploading files to the server, the files are automatically converted into Unix files with LF line endings. This is done by running all input and answer files through the `vim` [VIM] command `vim + "argdo set ff=unix | update" +wqa ./*.txt` in the directory containing the problem files. The checker source file is not run through the command as the g++ compiler handles both Unix and DOS filetypes.

Also, some errors occurred when administrators tried to remove locked submissions in the database through the admin interface. Due to lack of training, it often left the database and the server file system in an inconsistent state. As explained above, cascading delete was added between the `Run` and `Submission` tables to partially solve the problem, but the files associated with a submission would remain in the server file system and had to be removed manually. However, deletion of submission files was often forgotten, and it caused filename conflicts if the user re-submitted zip files with equal folder and file names as a previous submission.[5]

The Flask-Admin [FLAc] procedure `on_model_delete` solves the above inconsistency problem. It has been added to handle when submissions are deleted from the database and takes care of removing all files in the server file system associated with a deleted submission. However, it is still possible to leave the database in an inconsistent state if one does not use the admin interface to delete database content. If other tools than the Flask-Admin module are to be used in the future to handle the database, SQLALchemy provides event listeners that can be used instead of the above-mentioned procedure. As the CMB team currently uses the admin interface to handle database content, the `on_model_delete` procedure in the Flask-Admin interface is used.

Administration of the Bulletin table presented in Section 4.3.2 has also been added using the Flask-Admin module. Administration of bulletins is shown in Figure 4.15, which shows how to create bulletins and how to view stored database bulletins.

[5]A suggested future improvement is presented in Section 7.2, which is based upon storing submission files by the uniquely generated database id instead of using user defined folder and file names.



Content	This is a removable message!
Valid Until	2016-05-14 00:00:00
Removable	✓

		Content	Valid Until	Removable	Created
		This is a message!	2016-05-13 00:00:00	⊖	2016-05-12 11:26:16
		This is a removable message!	2016-05-14 00:00:00	✓	2016-05-12 11:27:07

Figure 4.15: Bulletin creation and overview in the admin interface.

4.4 Backend

Some small changes have been made in the backend code. The CMB team discovered that there was no timeout when running the small correctness test, which could potentially lock the backend if the submitted code contained infinite loops. Locking the backend is not acceptable under any condition, and the Unix program `timeout` [TIMa] is therefore used during the small correctness test to kill the submitted program after some time if it halts for too long. The big correctness test already uses `timeout` to kill programs that stall or are too slow, and the timeout is set to 90 seconds. Since the small input set is much smaller than the big correctness test, it is currently set to a third of the timeout of the big correctness.

The JSON returned by the backend is also slightly changed. A "state" field was added to track the condition of the program better as it executes on the backend, especially if there is an error during execution. The field is currently used as input to the "detailed_state" database field as described in Section 4.3, which is further used in the frontend to specialize the feedback given to the user if there is an error during execution of the program. Currently, the field is a simple string describing the state of the submitted program when the backend is done executing, but can be extended with for example JSON structures as more information about program state becomes available.

4.5 Improvement Proposals

4.5.1 Stability Test

An integration test[6] was developed during the specialization project in the Autumn of 2015. The project report further described the usage of the integration test in a stability test that determined the accuracy of execution time and energy measurements over multiple submissions. A single run of the integration test sets up a test server and submits a correct solution to a problem specified by the test, by using the `nose` test framework which is, as mentioned in Section 2.2.5, used to develop and run unit tests. The backend and server code was also temporarily modified to output temperature readings before and after the run. A simple Bash-script was

[6]Tests multiple units or components of a system in the same test.

created in order to run the test multiple times with varying delay between each run of the integration test.

Further extending the system with an improved stability test is wanted by the CMB team. This thesis proposes how to further extend the stability test, as stated in objective `P1` presented in Section 1.2. The objective aims to extend the stability test by simulating synthetic workload from users. Many test frameworks make it possible to simulate real users, but to restrict the discussion of frameworks we will only consider those that fit into the existing server code base, i.e., the test framework should be based on Python. Introducing frameworks dependent of other languages is not beneficial, as the CMB system is already fairly complex and it would require a lot from future developers. It is also a great advantage if the stability test could execute automatically, for example as part of a stage in a Jenkins pipeline or in a daily executed cron job.

The Python based framework Funcload [FUN] is a good alternative and is used by Mozilla Services among others.[7] The advantage of this framework is that tests are launched through a terminal, which makes it simple to automatically launch tests and benchmarks. The frameworks offer setup of benchmarks that can simulate multiple users or clients making requests against the server, and the requests made can also transport any kind of content such as zip files. This makes it possible for the simulated users to upload and execute submissions to a problem, as in the integration test described above. The framework can also generate HTML or PDF reports after a benchmark execution, containing detailed information about the test setup, request charts and network statistics, server CPU load and memory usage, as well as possible failed requests.

Another benefit in using the Funcload framework is that it requires small changes to the existing integration test. The functionality of the Bash-script, i.e., gathering data over multiple submissions, can be covered in tests written in the Funcload framework as it enables simulation of multiple clients. The server and backend code requires some changes in order to make the test gather data from multiple runs, and the development of the improved test should be done in correspondence with the implementation of gathering low-level statistics listed in Appendix C. Results and measurement statistics such as temperature during execution should be associated with submissions made by users, and should be stored in the server file system. Under the assumption that results and statistics from runs are stored in the file system, the improved stability test should also automatically report the stability of execution time, energy, and temperature readings to cover the functionality of the previous stability test.

Another framework alternative is Locust [LOC], which was also mentioned in the specialization project report. The downside of the framework is that it sets up a web server with a web based user interface, which requires us to start the test

[7]See blog post: `http://blog.ziade.org/2013/04/25/thoughts-on-load-testing/`.

manually through a browser. It is worth mentioning that it is possible to make the test execute through a terminal by requesting information from the Locust web-server using HTTP requests, but the framework will still set up a web-server, which seems to be a little drastic compared to our simple requirements. However, the web interface is simple to use and is also very popular on GitHub.

Both frameworks mentioned above require a running CMB server and backend to work. The previous integration test only required a running backend as the test automatically started a test server, and it should be explored wether it is possible to do the same with the introduction of one of the above frameworks to follow the testing standard in the project. Another alternative is to run the stability test on a designated server, which is a true copy of the production server, that is used by few real users. The introduction of a new designated production like test server is discussed further in Section 7.2 as possible future development.

4.5.2 How-To Page and About Page

The user feedback from TDT4200 [TDTb] pointed out that the HowTo page was unclear on how to structure the content of zip files to be uploaded. Some users also reported that it would be nice with an explanation of Unix error codes, which is presented to users if their submission terminates with an runtime error. In addition, a solution to the OS X upload bug was discovered during the Spring and has been implemented as explained in Section 4.2. However, if OS X users still experience problems, an explanation of how to correctly create a zip on OS X systems should be added to the HowTo page. As the system was to be used in experiments in the course TDT4102 [TDTa] and in the user test presented in Section 5.2, the HowTo page was updated with the above information.

The page should also be further expanded as the system is extended with more features. There are also some small possible extensions, which can be made to improve the content on the HowTo-page. Among the things missing are IO handling examples in the C programming language, as the current page only has an example of C++ code IO handling. The text explaining the uploads can also be further extended with a descriptive illustration to make it even more clear how to upload code to the current system. The HowTo could also contain an FAQ, which can be updated as questions are reported by users.

Another idea apart from the HowTo page is to add an About page. The About page should describe the project and system in an easy way, as well as describe the motivation and purpose. The page should also explain technical aspects, such as an explanation of EDP, which may be an unknown metric for programmers new to energy efficiency metrics. A list of publications related to the system would also be beneficial on this page. The About page was not added to the system in the project scope of this thesis.

4.5.3 Adding Problems

This thesis has not contributed towards adding new problems. However, all problems used in the CMB system are advised to follow the standard of problems present in other OJs. Some of the OJs mentioned in Section 2.3 are used in ICPC programming competitions, and seem to follow the same format in their provided problem descriptions. The CMB system should aim to offer problems following a similar format, as the system may be used in similar competitions in the future. Follan and Støa also described this in their Master Thesis [FS15]. The CMB team should also strive to add more problems, hopefully with as much diversity as offered by other OJs.

4.5.4 Discussion Forum

A discussion forum is wanted by the CMB team. There exists multiple third party solutions offering online forums that offer varying degrees of modifiability, and we here restrict the discussion of third party solutions by requiring that the software is related to technologies used in the system or is known by the CMB team. The framework FlaskBB [FLAg] is a Python Flask based lightweight forum module, and seems to offer some degree of modifiability. The framework offers simple forum functionality such as posting questions, issue tracking, private messages, and forum administration. It is also possible to create custom themes to better match the frontend view. There is also a popular Javascript based open-source forum software called Flarum [FLAb], which offers much the same features as FlaskBB. It is also worth mentioning that third party software like Piazza is an option, but future developers should keep in mind that the software offers only small modifications of its appearance.

A deeper study should be conducted before choosing how to implement a discussion forum. To make the system look more professional, a third party module with a highly modifiable appearance should be chosen. Another possibility is to build the forum software from scratch. A disadvantage is that it takes a lot of time to develop forum software, but a benefit is that no new software needs to be introduced in order to develop the forum and it makes the system look more professional to the outside world.

Chapter 5

User and System Testing

This chapter will explain the testing of usability of the developed system. Section 5.1 starts off with a brief presentation of the small continuous user testing, which was concucted during development of new features. Further, Section 5.2 presents the motivation, methodology, and results from a large scale user test conducted April 18th, and the results are compared to results gathered and presented in the specialization report. The chapter ends with a presentation of system unit test coverage in Section 5.3. This chapter covers the following objectives presented in Section 1.2:

- Conduct a user-experiment to evaluate system usability, i.e., objective `U3`.

5.1 Continuous User Testing

ISO 13407 part 210 [ISO99] describes the practice of user-centered design. To achieve high usability in a system, it is important to involve end-users early in the design process, as they might have other requirements to the system compared to system designers and developers. Figure 5.1 summarizes the iterative steps of the process. First we have an idea and specify the usage of the idea in the system, before specifying requirements set by users. Then we develop a design prototype, and evaluate whether the design meets user requirements. If the new design meets user requirements it is integrated into the system, but if it is not accepted, we go back to the stage in the user-centered design process, which needs revision or modifications.

System development during the Spring has followed a similar process as the user-centered design approach described above. Requirements were specified by the CMB team, and a design prototype was developed and tested on possible end-users. Users were asked if they liked the new feature and they were also presented with the old user interface for comparison. If the test users liked the new feature it was accepted, otherwise it was further developed or revised until accepted.

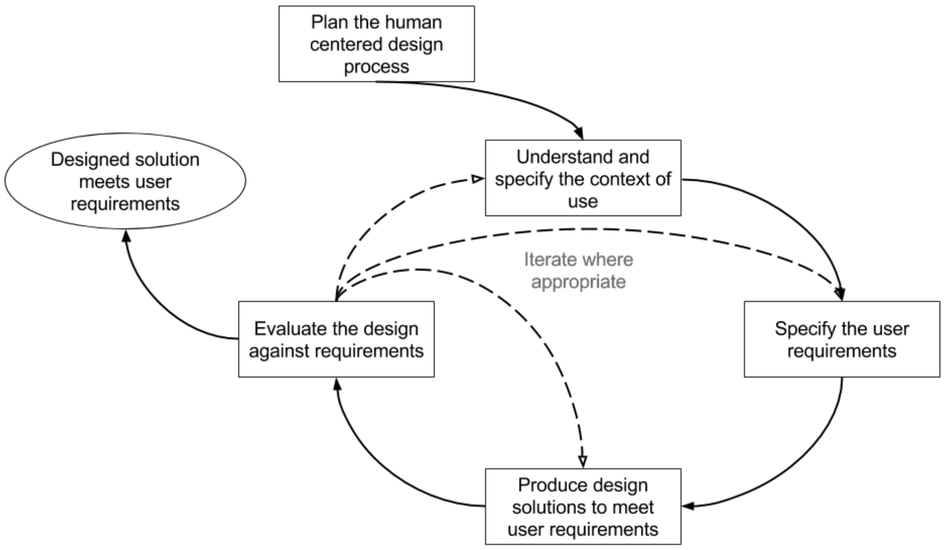

Plan the human centered design process
Designed solution meets user requirements
Understand and specify the context of use
Iterate where appropriate
Evaluate the design against requirements
Specify the user requirements
Produce design solutions to meet user requirements

Figure 5.1: User-centered design process (adapted from: [ISO99]).

Test users included supervisors, project team members, and student acquaintances. Both the admin interface and user interface features were tested according to the above process. Future developers are advised to follow a user-centered design approach such as the Lean Startup methodology [Rie11], which has become fairly popular during the last few years.

5.2 User Experiment

5.2.1 Motivation

A user test was performed in the specialization project and the project report was delivered in December 2015 as mentioned in Section 2.2.7. The CMB system was one of three tools used by the students in mandatory assignments, where the CMB system mainly was used for testing the system on more users then previously attempted by Follan and Støa [FS15]. The course assignments cover various topics and methods of producing parallel code in C++, like OpenMP, NEON, MPI and CUDA, and is about 25% of the workload during the semester for the average student. The TDT4200 course staff added in total five programming problems to CMB, which were solved by most students, and the system received a lot of submissions during the semester.

During November of 2015 a total of 37 students delivered an optional questionnaire, which included some questions about the CMB system. In addition to the questionnaire, some feedback on the system was gathered during the last lecture of the course late in November. The feedback was mainly focused on usability aspects of the system, which helped the CMB project team to discover bugs, and to setup and prioritize features to implement next. The prioritized list generated a backlog, which was used as a motivation when constructing the objectives presented in Section 1.2.

As this thesis potentially has improved system usability, it is interesting to conduct a user study to compare the usability of the two system versions. As mentioned in Section 2.2, the system developed by Follan and Støa will be referred to as system version one, while the updated system presented in Chapter 4 will be referred to as system version two in the below discussion. As mentioned in Section 3.1, usability is a broad term and this thesis has contributed mainly against improving efficiency, learnability, and satisfaction of users as discussed in Chapter 3 and implemented in Chapter 4. The above observation makes it interesting to test the following hypothesis:

Hypothesis 1: The usability, in terms of either efficiency, learnability, or user satisfaction, is higher in system version two compared to system version one.

5.2.2 Methodology

The user test was conducted April 18th this Spring and assesses the system improvements presented in Chapter 4 with focus on usability. The goal of this second test is to further improve the results received from the specialization project user test also called user test one. The second test required an interest in programming and programming experience from the participants, but no prior knowledge of C or C++ or any parallel programming libraries were set as a requirement to participate in the test.

The weak requirements were set since there was generally low interest from students in participating in the test. The low interest in the test is most likely due to bad timing, as many students have several assignment deliverables before their last lectures in mid April. Furthermore, attracting students or others to participate in a user test for a master thesis is generally hard if the test takes several hours. *As the focus of this test was to test the usability of the system, it should be sufficient for participants to have general knowledge and interest in programming to assess system usability.*

The user test described a set of tasks to be completed by each participant. Since we are comparing the two system versions, the test does not assess the new features implemented and is assumed covered by Section 5.1. Appendix A presents the tasks executed by every participant during the test, along with a questionnaire each participant was to complete before finishing the user test.

Problem Name	**Problem Description Summary**
Hello World	Print out ”Hello World!””.
Digits	T test cases are given as input, each test case describing the range between two numbers N and M. For each test case, output the number of zeroes present in all the numbers in the range $[N, M]$.
Prime Number	For every number in the input, output if the number is a prime.
WERTYU	For every encrypted input string, decrypt it and output the decrypted string. Hint: The input strings are encrypted with a QWERTY keyboard shifted one character to the right.
Reverse String	For every input string output the reversed string.

Table 5.1: Available solvable problems during user test.

Nearly all participants conducted the test simultaneously in a lecture hall to simulate the heavy submission traffic we experienced before assignment deadlines in TDT4200. A couple of participants started the user test late or participated a couple of days after the official user test, to simulate the students in TDT4200 that experienced little or no traffic. Participants were informed that questions related to how to complete a task would not be answered during the test and that tasks should be completed individually. They were also informed that they could abort the test at any time.

The most time-consuming task the participants had to complete was to submit code to one of the problems presented in Table 5.1. Submitting code to problems is the core of the OJ and the process of uploading and running submissions was the most common procedure done by TDT4200 students. The procedure and aspects related to submissions was also mentioned the most times in feedback given by the TDT4200 students, and was therefore chosen as one of the main tasks to perform in this user test. Before starting the user test, the participants were also notified to comment extensively on the procedure when filling out the survey.

The participants who were unfamiliar with C and C++ were given five different source files to the “Prime Number”-problem, which they were to submit to the system. The thought behind handing out source files, was to simulate that the participants made a number of tries before arriving at the correct solution. The “Prime Number”-problem was chosen as it is of medium to easy difficulty. Participants who knew either C or C++ were also motivated to try to solve more than one problem if they arrived at a solution quickly, in order to extensively test the

Question Type	**Likert Score Range**				
	1	2	3	4	5
A	Very poor	Poor	Neutral	Good	Very good
B	Strongly disagree	Disagree	Neither agree or disagree	Agree	Strongly agree
C	Very hard	Hard	Neutral	Easy	Very easy
D	Not satisfied at all	A little bit satisfied	Neutral	Satisfied	Very satisfied

Table 5.2: Likert scale alternatives on question type.

procedure of uploading and running submissions. Other tasks executed by participants included sign-up, login, viewing submissions and highscore lists, joining groups, creating and administering groups, and changing user e-mail and password.

The questionnaire found in Appendix A contains the same questions about the CMB system presented in the questionnaire given to the students in TDT4200. The questionnaire is, as mentioned in the Specialization project report, inspired by a typical Likert scale form developed by IBM[1] and complemented with textually based questions. To compare the results of the two questionnaires, it is important that they cover the same aspects, and all questions developed in the specialization project are therefore reused in the questionnaire handed out in this user study. In addition, a couple of extra multiple choice questions were added to the questionnaire of this test for the sake of clarity.

Likert scale questions are often used in usability assessment of software systems. In this test, the number of Likert scale alternatives is set to five per question. Each row in Table 5.2 shows the possible alternatives for a given question type, as well as each alternative's corresponding score ranged from one to five. The answers to Likert scale values of the two user tests are then compared using statistical analysis as explained in Section 5.2.3. To make it easier for participants to describe their thoughts about system features, the textually based questions are provided in the questionnaire and are further used as qualitative support when discussing the user study results.

5.2.3 Statistical Analysis

The tool Gnumeric [GNU] is used for statistical data analysis comparison of the multiple choice results of the two user tests. The results are reproducible using the pie charts and number of participants of each user test found in Appendix A. Each multiple choice answer from both tests is translated into the corresponding

[1]See: `http://garyperlman.com/quest/quest.cgi`.

Likert scale value, and values of matching questions between the two user tests are then compared. To simplify the discussion of results and validity, the user test conducted as part of the Specialization project will be referred to as user test one or T1 and the user test conducted in this thesis will be referred to as user test two or T2.

An F-test [Moo07] is conducted to determine whether the compared data set variances can be considered equal. The F-test calculates a P-value, which is a measurement of how extreme the results of a given test are relative to the underlying statistical model. The P-value is compared to a defined significance level, and variances are considered equal if the P-value is less than the decided significance level or unequal if the P-value is bigger than significance level. The result can also be formulated as keeping the null hypothesis (H_0), while the opposite situation is referred to as rejecting the null hypothesis and accepting the alternative hypothesis (H_1).

A T-test [WMMY93] is performed on each question to determine whether two user tests have significantly different means. The T-test can be executed either assuming equal or unequal variances, and the result of the F-test determines which version of the T-test to execute. The T-test also outputs a P-value, which is used in the same way as explained in the above paragraph.

The statistical analysis in this thesis assumes a significance level (α) of 5%. Furthermore, the null hypothesis (H_0) for the T-test assumes equal means of the two datasets, while the alternative hypothesis (H_1) is set depending on if we are conducting either a one-tailed or two-tailed T-test. Equation 5.1 shows the possible alternative hypothesis for the T-test, where μ_1 and μ_2 are the means for user test one and two respectively. In this thesis we use the one-tailed alternative hypothesis, as we want to obtain stronger results for system version two.

$$H_1 = \begin{cases} \mu_2 > \mu_1 & \text{if one-tailed test} \\ \mu_1 \neq \mu_2 & \text{if two-tailed test} \end{cases} \tag{5.1}$$

If the T-test indicates a significant difference between the two user groups further analysis is needed. If so, an Anderson-Darling test [RW11] is performed to check the normality of the data sets, as normality is often a requirement for a valid T-test. However, if the normality check fails, the non-parametric Wilcoxon-Mann-Whitney (WMW) test [HL05] is performed on each of the question groups accepted by the T-test. The test is used to support the conclusion of the T-test, as non-parametric tests are often used when the compared data sets are non-normally distributed.

The statistical effect-size is also reported and used when discussing the results. The effect-size is meant by the statistical *strength* of a result, and will be important in our discussion of the multiple choice results. The reported effect-size metrics used is Cohen's d, Hedge's g, and Pearson's r [Cum13].[2] Cohen's d is often used as

[2]Calculated with this tool: `http://www.polyu.edu.hk/mm/effectsizefaqs/calculator/`

Effect size	Pearson's r	Cohen's d	Hedge's g
Small	0.1	0.2	0.2
Medium	0.3	0.5	0.5
Large	0.5	0.8	0.8

Table 5.3: Effect sizes and corresponding metric values.

effect size to measure the strength of a T-test, as well as Pearson's r which is a well known effect-size metric. Hedge's g is also included in the results, as it is more accurate than Cohen's d with small sample sizes. Table 5.3 reports the strength scale for each of the used effect-size metrics, and a conclusion from a test is considered stronger the higher effect-size reported.

Cohen's d and Hedge's g are also used to calculate the statistical *power* of a conclusion. Statistical power is the probability of correctly rejecting H_0 when H_1 is true, and is important when discussing threats against validity to be sure that we have enough respondents to draw valid conclusions. Statistical power of the T-test is calculated using the Real Statistics Resource Pack in Microsoft Excel [RSR].

5.2.4 Results and Evaluation

There were in total 21 participants in the second user test. Table 5.4 shows a mapping between question titles and labels, which makes it easier to refer to the questions in the below discussion. As mentioned, there were a total of 37 TDT4200 students providing feedback in the optional questionnaire given during the Autumn semester. The pie-charts and a summary of textually based feedback for both the user test one and two can be found in Appendix A. All statistical test results reported in tables are rounded to three decimal places if applicable, that is, results are not rounded if it changes the outcome of the below discussions.

Table 5.5 shows the means and variances of the user test one and two responses for each Likert scale question in common for the two user tests. The table also include F-test P-values and the conclusion of the F-test compared to the significance level, which determines to either assume equal or unequal variances when performing the T-test. The T-test P-value is also reported, and is used to determine if a rejection of null hypothesis is needed.

The three questions `A3`, `D1`, and `A4` should reject H_0 as indicated by the T-test results in Table 5.5. However, none of the data sets are normally distributed according to the Anderson-Darling test. Subsequently, a WMW-test is conducted to support the conclusion of the T-test. Table 5.6 reports the resulting P-value of the WMW-test, as well as effect-sizes and power calculations.

`calculator.html`.

Label	**Question Title**
`O1`	"What year of study are you in?"
`O2`	"Which of the following best describes your study programme?"
`A1`	"How would you rate the Climbing Mont Blanc system in general?"
`B1`	"It was easy to use the system?"
`A2`	"How would you rate the usability of the Climbing Mont Blanc system?"
`C1`	"Was it difficult to learn how to use the Climbing Mont Blanc system?"
`A3`	"How would you rate the design of the Climbing Mont Blanc user interface?"
`C2`	"How would you rate the process uploading and running a program?"
`D1`	"How satisfied are you with the feedback given by the Climbing Mont Blanc system?"
`B2`	"The feedback given by the system is clear and helpful?"
`A4`	"How would you rate the information on the HowTo-page?"

Table 5.4: Label to question title mapping: Labels are created according to Table 5.2. Questions marked `O` are meant as other questions not within the five point Likert scale range.

The WMW-test also indicates significant results as its P-values are lower than α. Question `D1` has the most significant results, as the effect-size is large and the power is high using both Cohen's d and Hedge's g. Using Hedge's g to compute power, we can by a 97.2% confidence value be certain that we have correctly rejected H_0 when H_1 is true. We can therefore conclude that users, most probably, are more satisfied with the feedback of system version two. The results reported for question `B2` in Appendix A also indicates that feedback is good, as over 80% of participants either agree or strongly agree that the feedback given by the system is clear and helpful.

The questions `A3` and `A4` are also accepted by both the T-test and WMW-test. However, their strength in terms of all effect-size metrics indicates medium strength for `A3` and low to medium strength for `A4`. The statistical power of question `A3` and question `A4` should have been higher for us to draw any valid conclusions. We can only conclude that system version two might have a better user interface design and an improved HowTo-page.

The overall usability is good in both system versions. This also corresponds to the conclusion of tests ran against question `C1`, and answers to textually based feedback received from both user test one and user test two. There is also generally

Question	**A1**		**A2**		**C1**		**A3**	
	T1	T2	T1	T2	T1	T2	T1	T2
Mean	3.676	3.857	3.730	3.809	3.919	3.905	3.730	4.143
Variance	0.392	0.429	0.592	0.162	0.521	0.590	0.369	0.329
F P-value	0.396		0.002		0.362		0.400	
Variance	Equal		Unequal		Equal		Equal	
T P-value	0.151		0.303		0.472		0.002	
Reject H_0?	No		No		No		Yes	

Question	**C2**		**D1**		**A4**	
	T1	T2	T1	T2	T1	T2
Mean	3.595	3.524	2.568	3.619	3.324	3.842
Variance	0.970	0.762	0.919	0.947	0.892	0.474
F P-value	0.287		0.454		0.060	
Variance	Equal		Equal		Equal	
T P-Value	0.393		$9.6x10^{-5}$		0.022	
Reject H_0?	No		Yes		Yes	

Table 5.5: Mean, variance, F-test, and T-test results.

little overlap in textually based feedback, which indicates that there have been some improvements to usability. Some of the textually based feedback, such as adding language versions and project information to the HowTo-page as proposed in Section 4.5, has already been noted by the CMB team and is already listed in the backlog found in Appendix C.

The textually based questions are as mentioned meant as support for the multiple choice questions. The questions are also important for the future development of the CMB prototype. Proposals of new features, such as reporting low-level statistics, allowing uploading multiple files, and updated feedback on placement in the run-queue have been noted by the CMB team. Section 7.2 presents these features and they have also been added to the backlog in Appendix C.

Some of the feedback noticed by participants, such as real-time updates of submission placement in the run-queue, was actually in the making shortly before the conduction of the user experiment. As the improvements showed to be more extensive to implement than first thought, they were not implemented. The choice was made as it was little time to develop unit tests and test the implementation before the conduction of the user test, and the features also required changes to backend code which was the domain of another master student. Chapter 6 further discusses the matter.

The experiment results indicate that users are more satisfied with version two of the system. Users are from the above study more satisfied with the feedback given by system version two, and there is also a trend towards users being more satisfied

Question	**A3**	**D1**	**A4**
WMW P-value	4.231%	0.081%	4.400%
r	0.330	0.478	0.283
d	0.699	1.088	0.590
g	0.684	1.076	0.557
Power (using d)	0.711	0.975	0.564
Power (using g)	0.692	0.972	0.518

Table 5.6: Wilcoxon-Mann-Whitney P-values, effect-sizes, and power results.

with the user interface design and the information displayed on the HowTo-page. We can therefore conclude that it is likely that usability in terms of either efficiency, learnability, or user satisfaction is better in version two of the system, i.e Hypothesis 1 seems to be true.

Likert scale questions covering overall usability (`A2`) do not indicate a significant improvement in this experiment. Both test one and test two indicated good overall usability. However, users were only testing actions in common of the two system versions. New features implemented during the Spring were not included in this experiment, but are considered covered by the continuous user testing described in Section 5.1. Since the range of possible actions against the system has increased in version two and users has approved the new features, we can to some extent also argue that the overall usability has improved.

5.2.5 Threats to Validity

Conclusion validity concerns to what extent conclusions from statistical analysis are correct. It is closely related to statistical power, and is important when considering to reject H_0. In statistical analysis we typically have false negatives or false positives. False positives (called Type I errors) means that significant results are found even though data indicates otherwise, for example due to too few participants. Further, false negatives (called Type II error) can be present, which means that H_0 is not rejected even though there could have been an effect with a larger number of participants. In the below discussion we will assume we at least want a power of 80%, to be sure that a conclusion is correct.

Question `D1` seems to be a valid conclusion. We could actually have a power of 80% with only 20 participants in user test one and 12 participants in user test two to draw a valid conclusion from the results.[3] Question *A3* has too few participants to draw a valid conclusion, however there is a trend indicating a better scored satisfaction over the new user interface. We would have reached our target statistical power with 47 participants in user test one and 27 participants in user test two. However, satisfiability in system design is only one aspect of usability

[3]Calculated using this tool: `http://www.biomath.info/power/ttest.htm`.

and we also need to keep in mind that participants were only testing a subset of the new features in the system.

Question *A4* had statistically lowest power of the three questions having statistically significant results. The power at 51.7% using Hedge's g is not strong enough to draw a valid conclusion. We would have needed 70 participants in test one and 40 participants in test two to correctly accept H_0 with a power of 80%. However, improvements to the HowTo-page are listed as secondary improvements in the objectives listed in Section 1.2, and the small changes made presented in Section 4.5 were due to the usage of the system in a learning experiment.

Construct validity concerns to which degree an experiment actually measure what it declares to be measuring [CM55]. The user experiment tried to measure if there were any improvements to usability in system version two. As mentioned, some questions were added to the questionnaire this Spring, which could potentially damage the validity of results. However, these were only added for clarity and should not damage the conclusions made above. Also, participants might have had trouble assessing usability as they might have used a long time developing code instead. However, participants were given the option of receiving a set of programs simulating multiple tries against the system before arriving at the correct solution.

Internal validity concerns if there is a causal correspondence between the methodology used and the results of the experiment [Oat06]. If there are other factors, or confounding variables, in the experiment which lead us to the same conclusions, the measurements have poor internal validity. There are some factors in the above experimental setup that may threaten the internal validity of the results.

Participants in the second user test do not necessarily have the same background and interests as the participants of the first user test. This may have an effect on how the participants perceive the system, and it may be different from the perception made by students students in TDT4200. Also, some participants of the second user test were not familiar with C or C++, and received a set of executables for a programming problem to simulate normal system users. The goal of this user test was to test system usability and not programming skills or personal interests, and both groups of participants also had the same foundation when starting to use the system.

Participants in user test two also used the system for a short period of time compared to participants in user test one. The participants in user test two may not have been able to test the system thoroughly like the students in TDT4200 had a chance to do, as they used the system in a total of five exercises throughout the Autumn semester. Participants in user test one might, therefore, have more experience in using the system. However, participants in the second user test were adviced to test the system thoroughly if they finished early.

Many of the participants are friends of or familiar to the CMB team. They were invited due to low interest in the user experiment. Participants familiar to the system or CMB team might affect the above results further in either a positive or negative way. The participants were therefore kindly asked to give as objective feedback as possible. In order to strengthen the above statistical analysis more research can be conducted.

External validity concerns the generalizability of measurements and results [Oat06]. The participants who conducted the second user test were familiar to the CMB team, and some also had little interest in parallel programming or C/C++ programming. More research needs to be conducted in order to determine whether the improvements are generalizable to cover programmers interested in low-level parallel programming. It is also worth mentioning the difference in distribution of year of study, where most participants in user test two are 5th year students as apposed to 4th year students in user test one. A further discussion of possible test setups is presented in Section 6.2. Regardless of participants' background, the results of the experiment is interesting and we have hopefully made more programmers aware of heterogeneous programming and OJ systems.

5.3 System Unit Tests

This section will present the statement coverage of system unit tests. A high test coverage is important to ensure correct functionality, and allows quick detection of features that break system functionality during development. Unit tests were developed during implementation of the system improvements described in Chapter 4, and are reported to demonstrate the correctness of developed code and the resulting system. This thesis has a goal of 90% test coverage for the system as introduced to the project by Follan and Støa [FS15].

The reader should be aware that the unit tests developed by Follan and Støa are also included in the below presentation of coverage. This thesis has only developed or extended unit tests for the system improvements presented in Chapter 4. This section concerns the correctness of improved functionality in a working system, and reader should refer to the Master Thesis of Follan and Støa for an overview of previously developed functionality and in detail test coverage of that functionality.

Information about how to run system unit tests and generate coverage reports can be found in Appendix E.

5.3.1 Frontend

The frontend unit tests are developed in Jasmine [JAS] as mentioned in Section 2.2.5. Also, Karma [KAR] is used as test runner to develop the coverage

Directory	**Statements**	**Statements covered**	**Coverage**
config/	1	1	100%
controllers/	759	676	89%
directives/	78	9	12%
services/	33	19	58%
Total	871	705	81%

Table 5.7: Frontend total test coverage.

reports, and Gulp [GUL] is used to launch the Karma test runner.

Table 5.7 shows the coverage of all files for the frontend code. The overall test coverage is under 90% as set above. However, as mentioned by Follan and Støa, the low coverage is due to missing tests to third-party code in the *directives/* directory, and is only used to give colored feedback of password strength during sign up which is not vital for the overall functionality of the system.

The *services/* directory also has low code coverage and is below the 90% requirement. The low coverage is mainly due to the implemented bulletin functionality presented in Section 4.2. The unit test covering the bulletin functionality is not complete, as it would require us to expose private functions within the bulletin component in order to write a unit test with good coverage. Instead of exposing private functions just for the sake of the unit tests, the component was tested manually. It is also worth mentioning that the component is not crucial for overall system functionality.

Controller	**Statements**	**Statements covered**	**Coverage**
error_msg.js	21	21	100%
forgot_password.js	9	9	100%
group.js	63	58	92%
home.js	32	28	88%
leader.js	81	72	89%
leader_problem_stats.js	70	67	96%
leader_user_stats.js	45	42	93%
login.js	12	12	100%
logout	11	11	100%
navbar.js	3	3	100%
newgroup.js	17	17	100%
problem.js	323	265	82%
profile.js	54	53	98%
signup.js	18	18	100%
Total	759	676	89%

Table 5.8: Frontend controller test coverage.

The components containing most of the frontend functionality can be found in the *controllers/* directory. The coverage of 89% overall is a result of the 82% coverage of the *problem.js* controller. The missing statements to be covered by test is mainly event based code, such as file upload and Socket.io events, and require unit tests to trigger fake events. While triggering fake events can be done in unit tests, the callback function[4] of the event runs code covered by other unit tests and developing unit tests to cover event based code were therefore considered less important. By disregarding the coverage of event based code, the coverage of the frontend should be above the 90% requirement.

5.3.2 Server

Table 5.9 shows the unit test coverage for the server code. The coverage of the server code is at 90%, which is therefore an accepted level of coverage. However, the coverage can be further improved by extending by improving the coverage of modules *cmb_utils.helpers* and *cmb_utils.wrappers.*

The unit tests do not cover some of the methods performing OS calls in module *cmb_utils.helpers.* OS calls are often simulated (mocked) in unit tests, as OS calls often take some time to execute and we want unit tests to execute as fast as possible. As the unit tests mock away most functionality of OS calls, these unit tests had a low priority and were instead tested manually.

The module *cmb_utils.wrappers* contains checks to validate session tokens and has the lowest unit test coverage. The missing unit test needs to test exception states which can occur, and has not been implemented. These unit tests have not been a prioritized, as the functionality has been extensively tested by Follan and Støa, and is also tested automatically during normal use of the system.

It is also worth mentioning that the Flask-SocketIO event module is not covered by the unit tests. As testing of the module involves multiple components of the system, that is, both the server, frontend, and their interaction, the tests can instead be considered as integration tests. Integration testing has been performed manually both locally and on the CMB development server, and the module is thus not included in Table 5.9. However, future developers should consider adding automatic integration tests at some later point, to lower the amount of manual testing needed before deploying to production.

[4]A function passed as an argument to second function, which is called during execution of the second function.

Module	**Statements**	**Statements missing**	**Coverage**
admin.admin	150	28	81%
cmb_utils.helpers	86	17	80%
cmb_utils.mail	5	0	100%
cmb_utils.wrappers	50	15	70%
database.models	168	10	94%
routes.bulletin	21	0	100%
routes.groups	208	3	99%
routes.problems	21	12	95%
routes.submissions	150	14	91%
routes.users	93	3	97%
server	72	7	90%
Total	1024	98	90%

Table 5.9: Server modules test coverage.

Chapter 6

Discussion and Evaluation

This chapter will discuss several aspects of this thesis. Section 6.1 will discuss pros and cons of the real-time updates described in Section 4.1. The Section will also discuss planned immediate next steps which were not integrated into the system before the user test, due to limited time to verify correctness of the code. Furthermore, Section 6.2 presents alternative ways of conducting the user experiment presented in Chapter 5, and Section 6.3 will discuss aspects of the system testing executed as part of this thesis. Finally, we will evaluate in Section 6.4 if the goals set in Section 1.2 have been reached.

6.1 Improvements

6.1.1 Real Time Updates

Section 4.1 describes the implementation of real time updates of data models using Socket.io and WebSockets. However, the current use case of WebSockets in the CMB only sends events from the server to the frontend to notify users about submission state updates. Another technology called Server-Sent Events (SSE) [Hic09] also enables the server to send updates to clients automatically without the need for polling. SSE uses the HTTP protocol to push updates from the server to connected clients, i.e., it is not full-duplex as the WebSocket protocol.

The great benefit of SSE is that it does not introduce a new protocol to achieve real time updates. SSE is thereby considered more fit to applications which only need to push updates from the server to the connected clients. The downside of SSE is that it does not support the browser Internet Explorer (IE), which in our case is unacceptable. The user interface of CMB is web-based, and we do not want to restrict users to certain browsers or OSs. Since IE is one of the main browsers used world-wide, WebSockets with Socket.io is used instead of SSE.

A benefit of using Socket.io is that the framework automatically detects which protocol that is supported by a given client, as mentioned in Section 4.1. As the framework automatically selects the protocol suited for a client, users are not restricted to a specific browser or operating system in order to use the system. As the socket is a full-duplex communication channel, it also makes it possible to implement features which are not possible using SSE. For the CMB system, a online code editor with automatic syntax error highlighting or a messaging service is possible using the Socket.io framework. Appendix C and Section 7.2 mention possible future extensions using the Socket.io framework.

The server uses the modules gevent [GEVa] and gevent-websocket [GEVb] as mentioned in Section 4.1.2. However, as mentioned on the documentation website of Flask-SocketIO creator Miguel Grinberg, it is also possible to use networking library eventlet [EVE] instead of gevent when using the Flask-SocketIO module [FLAf], and is reported to be the best performing option in combination with the module. There are however some benefits of using gevent, as it is tested in real-world high-scale environments and the module interface also follows Python standard library conventions.[1] The gevent module is therefore used in the CMB system.

6.1.2 Frontend

During development, we also planned to enable upload of single and multiple source files. The feature did not have a high priority for the CMB team at the start of the thesis, as feedback given by students in TDT4200 indicated that they quickly learned how to submit files to the system using zip-files. However, as indicated by the textual feedback from the user experiment conducted as part of this thesis, the feature is wanted by users, and as a result it has been added to the backlog found in Appendix C.

If submission timeouts also were added at the server (explained in Section 6.1.3), the problem-view also had a planned extension of dislaying a progress bar during execution. The progress bar would display the approximate time of execution, but the feature was not implemented as the extension was more complex than what it seemed on first glance. The feature has been added to the backlog in Appendix C as a usability improvement.

6.1.3 Server

A couple of server improvements were also considered during development but ended up with lower priority compared to the tasks listed in Section 1.2. First, the server should perform a simple check to verify the format of the uploaded zip-file. The frontend currently checks and does simple corrections to the zip file before sending it of to the server, as described in Section 4.2.1. Since the frontend is the main user interface of the system, zip-files submitted by normal users are therefore checked before sent to the server. However, if the system is to be extended with

[1]For a further discussion on the matter, see: `https://blog.gevent.org/2010/02/27/why-gevent/`, `https://groups.google.com/forum/#!topic/gevent/TelwPl3KgnE`.

other user interfaces, for instance a CLI, it would be beneficial to add server side zip-file checks.

During development and maintenance of the system there occurred file-name conflicts when storing submissions in the file system, as submissions are stored by submission name. The situation occured frequently during development of the submission delete endpoint described in Section 4.3.3. However, to avoid such file-name conflicts in the future, it could be an idea to instead save submission files by the automatically generated database id instead since it's unique. The fix was not implemented due to time limitations before the user test, but is added to the backlog in Appendix C.

Reporting the submission run-queue position to users was also planned before the user test. However, the extension turned out to be more complicated than at first glance. The current run-queue[2] is thread-safe and this is also required, as multiple users might access the queue simultaneously. However, there is no way of looking at elements and their position in the currently used queue module without removing them.

A simple solution is to copy the queue and emit its data over Socket.io to each of the connected clients every time a submission is pulled from the queue. If a client has a submission in the queue, the client could then simply loop through the queue and notify users of the new submission position. The solution is probably the simplest to implement, but would possibly impose transportation of unnecessary data to inactive clients. The solution would also increase the amount of network traffic during heavy system load, especially if the system is to be scaled with multiple boards and submissions are rapidly pulled out of the run-queue. As this solution was not discussed with the CMB team and there was little time to test the solution before the user experiment, it has not been implemented. The feature can be found in the backlog in Appendix C.

Timeouts were added to the backend to abort submissions which locked the backend for further use as described in Section 4.4. However, submissions could in theory crash from errors currently not handled by the system backend, or might be delayed due to high network traffic. The server should in such cases keep track of timers for each submission, and abort execution of a submission on the backend if a timeout occurs. The server could for instance fork off a gevent coroutine for each submission, and have each coroutine keep track of a timer for a given submission. Upon timeout, the coroutine could then kill the executing program on the backend and update the database with timeout information.

This feature was planned prior to the user test but not implemented. First, there was limited time before the user test to implement and test the feature. Second, developing low-level server and backend code was the focus of the Master Thesis

[2]Uses the Python Queue module: `https://docs.python.org/2/library/queue.html`.

written by Christian Chavez. To not interfere with the work done on scalability, the feature was given lower priority in this thesis and has been added to the backlog in Appendix C.

6.1.4 Backend

Debugging of submissions running over SSH has in some situations been troublesome as mentioned in Section 2.2.7. The CMB team therefore wanted to rewrite the scripts present at the backend into Python scripts instead which makes the scripts easy to unit test. As scalability and code development related to backend functionality were the focus of Master student Christian Chavez, the porting of bash scripts into Python scripts is not considered in this thesis. Only small changes were made to the backend as described in Section 4.4, to improve feedback to users in case of submission failures.

6.2 User Testing

The user experiment methodology presented in Section 5.2.2 corresponds closely to a static group comparison described by Oates [Oat06]. The static group comparison divides the participants into two groups, where one of the groups receives some form of treatment (version two of the system) and the other receives no treatment (version one). The effect of the treatment can therefore be assessed by evaluating test scores. There are some downsides with the method, such as in our case, we know that the group testing system version one had used the system longer and also had in interest in parallel C/C++ programming compared to the other group. As noted in Section 5.2.5, the difference between the two groups might have an affect on the results of the user experiment.

Oates also describes other common user experiment setups which could have been used in this thesis. Instead of regarding previous user test results, we could have tested system version one and two on the participants on the user test conducted this Spring only. Participants would then test system version one first and then system version two afterwards, which is known as a one group pre-test and post-test. The usability could then be assessed by comparing pre- and post-test scores. The downside with the method, is that participants might have learned from using system version one and it might affect the results when they are testing system version two.

Pre- and post-tests could also have been conducted if we had more participants to the user test conducted in this thesis. Participants would then be split into two random groups, each assessing the usability of system version one. If the randomization has been performed correctly, each group should have as equal assessment of usability of system version one as possible. The test is then run one more time, having one of the groups assessing the usability of system version two instead. The results are then compared, and differences in results of the two assessments are assumed to be caused by the different treatment of the groups. This method also has the same downside as described above: participants might learn from the first round of the user test and use their knowledge when assessing the system a second time.

There exists more experiment designs if a lot of participants are registered, such as the Solomon four-group design [Oat06]. However, as mentioned in Section 5.2.2, there were too few participants to consider using more complex methodologies. The static group comparison was chosen as there were few participants to the second user test, and because the Specialization project already had conducted a usability study. The benefit with the chosen methodology is that a limited number of resources is required. The methodology used only required one server hosting the new system, while the other user testing approaches described in this section requires two; one hosting system version one and another hosting system version two.

Most of the questions in the usability questionnaire used were made to a user study conducted during the specialization project. However, a common usability questionnaire like a SUS [B+96] could also have been used and may have been easier to analyze and validate. As there were few participants to the user experiment conducted in this thesis, we were forced to use the questionnaire constructed as part of the specialization project to correctly compare the results of the two user tests. But, the questionnaire is as mentioned based on a usability questionnaire developed by IBM and is also inspired by the questionnaire guidelines defined by Oates [Oat06].

litative analysis should also be put under consideration in the future. Quantitative usability studies mostly measure satisfiability of users, and we cannot verify that users have executed the tasks of the usability test correctly [Hol05]. A structured qualitative analysis should be conducted to validate other aspects of usability. However, the continuous user testing conducted, as described in Section 5.1, partially covers qualitative measures of usability.

6.3 System Testing

During this thesis several unit tests have been developed and their coverage were presented in Section 5.3. Also, manual testing has been conducted locally and on the development server of CMB. To speed up local manual testing, future developers should consider adding database fixtures[3] to the server code repositories. This would make it easy to load wanted data into the database before manually testing the system. Future developers should also continue to create unit tests and consider adding automatic integration tests to lower the amount of manual testing needed to accept a feature.

[3]Database fixtures are defined sets of test data which can be loaded into the database.

6.4 Project Objective Achievements

This section will evaluate whether we have reached the objectives defined in Section 1.2.

Main Objectives:

U1 - Fix the main bugs and known issues found during user testing of CMB in November 2015: Considered covered by Section 4.2.1. The section described how fixes of Mac OS X uploads (`U1.1`), locked submissions on backend (`U1.2`), and the highscore list sorting bug (`U1.2`) were implemented.

I1 - Change the existing database management system if necessary: Considered covered by Section 4.3.1.The SQLite DBMS were replaced by the MySQL DBMS, and all data present in the SQLite databases were transferred to the new MySQL databases for both the development and production server.

U2 - Improve and extend the CMB system's usability features in accordance with the CMB team's priorities: Considered covered by this thesis. Section 4.2.2 described the improvements done to feedback messages reported in the system, as well as upgrades done to the frontend views i.e `U2.1` and `U2.5` respectively. The improvement of adding and removing problems through the admin interface is covered by Section 4.3.4. The bulletin board extension to cover goal `U2.4` was described in Section 4.2.2. Section 4.1 described the implementation of real-time model updates with Socket.io (`U2.2`), which also has a lot of potential for the further development of the system. Some possible extensions are presented in Section 7.2 and is also listed in Appendix C.

U3 - Conduct a user-experiment to evaluate system usability: Considered covered by Section 5.2. A user test was conducted to evaluate system usability with focus on efficiency, learnability and satisfiability. As the user experiment used much the same questionnaire used in the user study conducted during the specialization project, a statistical analysis was conducted to compare the possibly improved system to the system developed by Follan and Støa.

Secondary Objectives:

P1 - Propose improvements to the existing stability test to simulating users and their submissions: Considered covered by Section 4.5.1. The Section described two possible Python modules which could be used to extend the stability test developed during the Specialization project. The proposed improvement have been added to the backlog found in Appendix C.

P2 - Propose how to improve the how-to information, the problems offered by CMB, and add new problems: Considered partially covered by this thesis. The improvements to the how-to information were discussed in Section 4.5.2, and the proposed improvements have been added to the backlog in Appendix C. This thesis has not contributed towards adding new problems, but Section 4.5.3 proposed that newly added problems use the same format as other OJs.

P3 - Propose how to implement a discussion forum: Considered covered by Section 4.5.4. The section proposed various third-party forum software packages which can be used by the system. Also, the section proposed that future developers should consider developing a discussion forum from scratch, to make the system look more professional.

I2 Implement some of the proposed solutions after approval by, and in collaboration with the CMB team: Not considered covered by this thesis. All proposals are currently listed in the backlog found in Appendix C.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis has improved and added more features to the CMB system. The features are mainly usability features with a focus on efficiency, learnability, and satisfiability. As usability is a broad term and as there is always room for usability improvements in a software system, only a sub-set of the usability improvements prioritized prior to the start of the thesis and urgent usability aspects discovered throughout this Spring have been implemented. Features implemented in the frontend, server, and backend source code have been described in detail.

The frontend views have also had some renovation, such as the removal of unnecessary information, structuring of reported information, and updating components with symbols. The improvements aim to make it easier for users to find important information, to navigate easier, and to more clearly state the outcome of interacting with system components. Feedback messages presented to users have also changed in system version two. Users are now presented with colored feedback messages when interacting with the system to make it easier to differentiate between various feedback. Submission errors are also displayed in a pop-up message instead of in a designated view, which removes unnecessary navigation stages. A spinner and real-time update of a running submission's state are also implemented in the system.

Real-time notifications with Socket.io was integrated into the system to enable automatic state updates of submissions. Further use cases of Socket.io and implementation suggestions of these features have briefly been presented. The potential of Socket.io in the system is huge and enables future developers to implement interesting features.

Furthermore, solutions to fixes such as enabling uploads for Mac OS X users, automatic Unix file format conversion, cascading-delete of submissions, and cancelation of submissions locking the backend have been described. The DBMS used has been changed from SQLite to MySQL, and all previous data present at the development and production server of CMB have successfully been moved into the new databases.

A user experiment has been conducted to evaluate system usability. The user experiment compared version one developed by Follan and Støa, with system version two developed in this thesis. The results shows that users are, by a 97.2% confidence value, more satisfied with feedback given in system version two. There is also a noticeable trend that users are more satisfied with the design and HowTo-page of system version two. However, we can not conclude with a confidence value of 80% or above that users are more satisfied with the design and HowTo-page in system version two. Improvements not tested as part of the user experiment, has been covered using continuous user testing conducted throughout the semester.

In conclusion, this project has contributed towards improving system usability and features of the CMB system. It has also contributed with proposals on features and implementation details for the future development of the system. The contributions are valuable for the future of the CMB system.

7.2 Future Work

This section describes possible future extensions which can be made to the system and the project in general. Each extension has a suggested ordering based on priority, and are marked either as A(high), B(medium), or C(low) priority improvements.

Project Administration

A. Problem descriptions should follow ACM ICPC standards: As mentioned in Section 4.5.3, problem descriptions should be in the same format as problem descriptions presented by other OJs. Follan and Støa also mentioned best practices when adding new problems to the system, and future administrators should follow them. Best practices and a manual for adding problems can be found in Appendix D.

A. Thorough testing of added problems: Problems descriptions should be thoroughly tested on the development server of CMB before adding the problem to the production server. This means that the input and output files of a problem to be added should be heavily tested to make sure it covers all edge cases present in the problem description. It is also important that substantial testing is performed either locally or at the development server, as we do not want users to be denied use of the system because of testing.

Development and Testing

B. Extra production server for acceptance testing: As described in Section 4.5.1, a production-like server should be added to more quickly run extensive testing and generate statistics in a production-like environment. The Jenkins pipeline stage should also be added to launch stability and integration tests automatically. The server should be added as it is considered a bad CI practice to run tests on the production server, as it may prohibit regular users from using the system.

C. Add Database Fixtures: Database fixtures should be added to the system. A fixture is a defined set of data which can be loaded into the database with ease, which enables quick setup before testing the system. Fixtures might make it easier for future developers to start developing, and run tests quickly during local development. A possible Python module which can be used is Flask-Fixtures [FLAd], which enables fixture definitions in the JSON format among others.

Features and Improvements

A. Server Side Timeout and Backend Monitoring: The server should keep track of timers for each submission running in the system as described in Section 6.1.3. The server can keep track of timers for each submission by for example using gevent coroutines [GEVa] as discussed in the section. The server could then kill submissions running on a backend if a timeout occurs and notify users by emitting events to the respective client who submitted the code. This can be achieved by exploiting Socket.io namespaces and rooms as presented in Section 4.1. Timeout and run-time information can also be sent in real-time to clients to enable users to have up-to-date information about a running submission.

A. Port Bash Scripts Into Python Scripts: Bash scripts defined at the server and backend should be ported to Python scripts instead. Debugging Bash scripts executed over SSH has proven to be troublesome, and have also proven to be error prone upon code changes. By rewriting all Bash scripts to Python scripts, it should be easier to debug, and it is also simpler to add unit tests for these parts of the system. The porting of Bash scripts into Python scripts should also make the system more stable, as the scripts has proven to be one of the bottlenecks in the system.

A. Server Side Zip File Check: The frontend parses the submitted zip and checks for the correct format as described in Section 4.2.1. A similar check should be added to the server code to ensure proper storage of files in the file-system, and is especially important if API documentation is released or a CLI is implemented.

B. Real-Time Queue Position: The server should give real-time information about submission position to clients who has submissions in the run queue. As described in Section 6.1.3, a complete copy of the queue can be made on every dequeue and the placement of every submission can be emitted using Socket.io. As discussed in the section, this solution might yield high network traffic during heavy system load, and it might be too simple. Another solution is to make a complete copy still, but only send submission positions to those clients with submissions in the queue. Socket.io rooms give a solution to the latter approach.

B. Multi File Upload: It should be possible to upload source files directly and preferably multiple files. More advanced upload feature such as an online code editor, which is offered by OJs like [KAT] and HackerEarth [HACa], should also be considered by future developers.

C. Low-level Statistics: The server should keep track of low-level submission information, such as memory usage, chip temperature, and cache usage. Further, the accuracy of running time and energy consumption measurements should by reported as well if applicable. This feature requires changes to all parts of the system.

C. Login via Feide: It should be possible for NTNU students to login via Feide. [1] Enabling login through Feide would make it simple for students and professors to start using the system in course-related activities.

[1]Feide is a student's digital identity on the Uninett network: `https://www.feide.no/`.

Bibliography

[A2O] A2 Online Judge Website. https://a2oj.com/. Last accessed 2nd of May 2016.

[ABL] ARM big.LITTLE Technology. http://www.arm.com/products/processors/technologies/biglittleprocessing.php. Last accessed 21st of April 2016.

[Ale14a] Alex Ramirez. *Building supercomputers from embedded technologies.* https://www.montblanc-project.eu/sites/default/files/publications/20140521mont-blanc-pracedays14.pdf, 2014.

[Ale14b] Alex Ramirez. *The Mont-Blanc prototype.* https://www.montblanc-project.eu/sites/default/files/publications/20140522montblanc-prototypes-workshop.pdf, 2014.

[ANG] AngularJS. https://angularjs.org/. Last accessed 14th of April 2016.

[B$^+$96] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[BFT13] Jean Luca Bez, Carlos E Ferreira, and Neilor A Tonin. URI Online Judge Academic: A Tool for Professors. In *PROCEEDINGS OF THE 2013 INTERNATIONAL CONFERENCE ON ADVANCED ICT AND EDUCATION*, volume 33, pages 763–766, 2013.

[BIT] Bitbucket Website. https://bitbucket.org/. Last accessed 14th of April 2016.

[BOW] Bower Package Manager. http://bower.io/. Last accessed 21st of April 2016.

[BSC] Barcelona Supercomputer Center. https://www.bsc.es/. Last accessed 12th of April 2016.

[BUL] Bull Atos Technologies. http://www.bull.com/. Last accessed 12th of April 2016.

[CM55] Lee J. Cronbach and Paul E. Meehl. Construct validity in psychological tests. *Psychological Bulletin*, 52(4):281–302, 1955.

[CN13] J. M. Cebrián and L. Natvig. Temperature effects on on-chip energy measurements. In *Green Computing Conference (IGCC), 2013 International*, pages 1–6, June 2013.

[COD] CodeChef Website. `https://www.codechef.com/`. Last accessed 2nd of May 2016.

[CRO] Crowsourcing definition. `http://www.crowdsourcing.com/`. Last accessed 2nd of May 2016.

[Cum13] Geoff Cumming. *Understanding the new statistics: Effect sizes, confidence intervals, and meta-analysis.* Routledge, 2013.

[DIF] Diff program. `http://man7.org/linux/man-pages/man1/diff.1.html`. Last accessed 20th of April 2016.

[EKN+11] Emma Enstrom, Gunnar Kreitz, Fredrik Niemela, Pehr Soderman, and Viggo Kann. Five years with kattis — Using an automated assessment system in teaching. In *Proceedings of the 41st Frontiers in Education Conference - FIE'11*, pages T3J–1–T3J–6, 2011.

[EVE] Eventlet website. `http://eventlet.net/`. Last accessed 30th of May 2016.

[EXY] Samsung Exynos Processors. `http://www.samsung.com/semiconductor/minisite/Exynos/w/`. Last accessed 8th of February 2016.

[FAI] Fail2Ban Website. `http://www.fail2ban.org/wiki/index.php/Main_Page`. Last accessed 26th of April 2016.

[Fil14] Filippo Mantovani. *Pedraforca: ARM + GPU prototype.* `https://www.montblanc-project.eu/sites/default/files/publications/workshopProto-Pedraforca.pdf`, 2014.

[Fil15] Filippo Mantovani. *High Performance Computing Based on Mobile Embedded Technology.* `https://www.montblanc-project.eu/sites/default/files/publications/Mont-Blanc-EMiT15-lq-public.pdf`, 2015.

[FJWC01] Xavier Ferré, Natalia Juristo, Helmut Windl, and Larry Constantine. Usability basics for software developers. *IEEE software*, 18(1):22, 2001.

[FLAa] Flake8 Documentation. `https://flake8.readthedocs.org/en/latest/`. Last accessed 21st of April 2016.

[FLAb] Flarum website. http://flarum.org/. Last accessed 20th of May 2016.

[FLAc] Flask-admin documentation. http://flask-admin.readthedocs.io/en/latest/. Last accessed 9th of May 2016.

[FLAd] Flask-fixtures website. https://pypi.python.org/pypi/Flask-Fixtures/0.3.7. Last accessed 8th of June 2016.

[FLAe] Flask-Migrate Documentation. http://flask-migrate.readthedocs.io/en/latest/. Last accessed 10th of May 2016.

[FLAf] Flask-SocketIO Documentation. http://flask-socketio.readthedocs.io/en/latest/. Last accessed 7th of May 2016.

[FLAg] Flaskbb website. https://forums.flaskbb.org/. Last accessed 20th of May 2016.

[FLAh] Python Flask. http://flask.pocoo.org/. Last accessed 20th of April 2016.

[FM11] Ian Fette and Alexey Melnikov. The websocket protocol. 2011.

[For13] Brian T. Ford. Angular socket.io component repository. https://github.com/btford/angular-socket-io, 2013. Last updated December 2014. Last accessed 8th of May 2016.

[Fow06] Martin Fowler. Continuous Integration. *Integration The Vlsi Journal*, 26(1):1–6, 2006.

[FS15] Torbjørn Follan and Simen Støa. Climbing Mont Blanc: A Prototype System for Online Energy Efficiency Based Programming Competitions on ARM Platforms. Master's thesis, Norwegian University of Science and Technology, Norway, 2015.

[FT00] R.T. Fielding and R.N. Taylor. Principled design of the modern Web architecture. *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, 2(2):115–150, 2000.

[FUN] Funcload documentation. http://funkload.nuxeo.org/. Last accessed 18th of May 2016.

[GEVa] Gevent website. http://www.gevent.org/. Last accessed 8th of May 2016.

[GEVb] Gevent websocket repository. https://bitbucket.org/noppo/gevent-websocket. Last accessed 8th of May 2016.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. In *Elements*, volume 47, pages 1–429, 1995.

[GIT] Git Version Control. https://git-scm.com/. Last accessed 14th of April 2016.

[GNU] Gnumeric website. http://www.gnumeric.org/. Last accessed 24th of May 2016.

[GRE] Green 500. http://www.green500.org/. Last accessed 12th of April 2016.

[GUL] Gulp Build System. http://gulpjs.com/. Last accessed 21st of April 2016.

[GUN] Gunicorn. http://gunicorn.org/. Last accessed 20th of April 2016.

[HACa] HackerEarth Website. https://www.hackerearth.com. Last accessed 2nd of May 2016.

[HACb] HackerRank Website. https://www.hackerrank.com/. Last accessed 2nd of May 2016.

[Hic09] Ian Hickson. Server-sent events. *W3C Working Draft WD-eventsource-20091222, latest version available at¡ http://www. w3. org/TR/eventsource*, 2009.

[HIG94] Mark Horowitz, Thomas Indermaur, and Ricardo Gonzalez. Low-Power Digital Design. *Lpe*, pages 8–11, 1994.

[HL05] J Hodges and E Lehmann. *Basic Concepts of Probability and Statistics*. Society for Industrial and Applied Mathematics, second edition, 2005.

[Hol05] Andreas Holzinger. Usability engineering methods for software developers. *Communications of the ACM*, 48(1):71–74, 2005.

[ICP] ICPC Website. https://icpc.baylor.edu/. Last accessed 27th of April 2016.

[IDIa] Idi department website. http://www.ntnu.edu/idi. Last accessed 13th of June 2016.

[IDIb] IDIOpen Website. https://idiopen.idi.ntnu.no. Last accessed 27th of April 2016.

[ISO98] W ISO. 9241-11. ergonomic requirements for office work with visual display terminals (vdts). *The international organization for standardization*, 45, 1998.

[ISO99] ISO13407 ISO. 13407: Human-centred design processes for interactive systems. *Geneva: ISO*, 1999.

[JAS] Jasmine Website. http://jasmine.github.io/. Last accessed 21st of April 2016.

[Jef13] Brian Jeff. big.LITTLE Technology: Moves Towards Fully Heterogeneous Global Task Scheduling. pages 1–13, 2013.

[JEN] Jenkins Website. `https://jenkins.io/`. Last accessed 21st of April 2016.

[JSH] JSHint Website. `http://jshint.com/`. Last accessed 21st of April 2016.

[JSO] JSON Data Format. `http://www.json.org/`. Last accessed 20th of April 2016.

[KAR] Karma Website. `http://karma-runner.github.io/0.13/index.html`. Last accessed 21st of April 2016.

[KAT] Kattis. `http://www.kattis.com/`. Last accessed 2nd of May 2016.

[KLC01] Andy Kurnia, Andrew Lim, and Brenda Cheang. Online Judge. *Computers & Education*, 36:299–315, 2001.

[LEE] LeetCode Website. `https://leetcode.com/`. Last accessed 7th of May 2016.

[LIG] Lightdm website. `https://wiki.ubuntu.com/LightDM`. Last accessed 14th of June 2016.

[LNAM12] Hallgeir Lien, Lasse Natvig, Abdullah Al Hasib, and Jan Christian Meyer. *ICT as Key Technology against Global Warming: Second International Conference, ICT-GLOW 2012, Vienna, Austria, September 6, 2012. Proceedings*, chapter Case Studies of Multi-core Energy Efficiency in Task Based Programs, pages 44–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[LOC] Locust repository. `https://github.com/locustio/locust`. Last accessed 18th of May 2016.

[MAL] *Mali OpenCL SDK download webpage.* `http://malideveloper.arm.com/resources/sdks/mali-opencl-sdk/`. Last accessed 13th of Juni 2016.

[MB] The Mont Blanc Project. `https://www.montblanc-project.eu/`. Last accessed 3rd of April 2016.

[MG97] G. De Michell and R. K. Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, Mar 1997.

[Moo07] David S Moore. *The basic practice of statistics*, volume 2. WH Freeman New York, 2007.

[NFS+15] Lasse Natvig, Torbjørn Follan, Simen Støa, Sindre Magnussen, and Antonio García-Guirado. Climbing Mont Blanc - A Training Site for Energy Efficient Programming on Heterogeneous Multicore Processors. *CoRR*, abs/1511.02240, September 2015.

[NGI] Nginx. `http://nginx.org/en/`. Last accessed 20th of April 2016.

[Nor88] Donald A Norman. *The design of everyday things.* Basic books, 1988.

[NOS] Nose Documentation. `http://nose.readthedocs.org/en/latest/`. Last accessed 21st of April 2016.

[NPM] npm Package Manager. `https://www.npmjs.com/`. Last accessed 21st of April 2016.

[Oat06] Briony J Oates. *Researching Information Systems and Computing*, volume 37. 2006.

[ODR] Odroid xu3 website. `https://forums.flaskbb.org/`. Last accessed 14th of June 2016.

[OEM] Odroid EnergyMonitor git repository. `https://github.com/hardkernel/EnergyMonitor`. Last accessed 21st of April 2016.

[OMP] The OmpSs Programming Model. `https://pm.bsc.es/ompss`. Last accessed 12th of April 2016.

[PBAL13] J. Planas, R. M. Badia, E. Ayguadé, and J. Labarta. Self-adaptive ompss tasks in heterogeneous environments. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 138–149, May 2013.

[PIP] Pip Documentation. `http://pip.readthedocs.org/en/stable/installing/`. Last accessed 26th of Oktober 2016.

[PKU] PKU JudgeOnline. `http://poj.org/`. Last accessed 2nd of May 2016.

[POS] Postman website. `https://www.getpostman.com/`. Last accessed 12th of May 2016.

[Rai13] Rohit Rai. *Socket.io Real-time Web Application Development.* Packt Publishing Ltd, 2013.

[RCG+13] Nikola Rajovic, Paul M. Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. Supercomputing with commodity CPUs: Are Mobile SoCs Ready for HPC? In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13*, pages 1–12, 2013.

[REC] RecSys Website. `https://recsys.acm.org/`. Last accessed 3rd of May 2016.

[Rie11] Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses.* Crown Books, 2011.

[RML08] Miguel A Revilla, Shahriar Manzoor, and Rujia Liu. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, 2:131–148, 2008.

[RRP$^+$13] Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. Tibidabo : Making the case for an ARM-based HPC system, 2013.

[RSR] Real statistic resource pack: Power of the t-test. `http://www.real-statistics.com/students-t-distribution/statistical-power-of-the-t-tests/`. Last accessed 25th of May 2016.

[RW11] Nornadiah Mohd Razali and Yap Bee Wah. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of Statistical Modeling and Analytics*, 2(1):21–33, 2011.

[SOC] Socket.io website. `http://socket.io/`. Last accessed 6th of May 2016.

[SPH] Sphere Website. `http://www.spoj.com/`. Last accessed 26th of Oktober 2016.

[SQLa] SQLAlchemy. `http://www.sqlalchemy.org/`. Last accessed 20th of April 2016.

[SQLb] Sqlite. `https://www.sqlite.org/`. Last accessed 20th of April 2016.

[STR] Stress command. `http://linux.die.net/man/1/stress`. Last accessed 21st of April 2016.

[TDTa] TDT4102 Procedural and Object-Oriented Programming Website. `http://www.ntnu.edu/studies/courses/TDT4102#tab=omEmnet`. Last accessed 8th of May 2016.

[TDTb] TDT4200 Parallel Computing Website. `http://www.ntnu.edu/studies/courses/TDT4200#tab=omEmnet`. Last accessed 27th of April 2016.

[TIMa] Timeout manual. `http://man7.org/linux/man-pages/man1/timeout.1.html`. Last accessed 8th of May 2016.

[TIMb] Timus Online Judge Website. `http://acm.timus.ru/`. Last accessed 2nd of May 2016.

[TOPa] Top 500. `http://www.top500.org/`. Last accessed 3rd of April 2016.

[TOPb] TopCoder Website. `https://www.topcoder.com/`. Last accessed 3rd of May 2016.

[UFW] Uncomplicated FireWall Ubuntu Documentation. `https://help.ubuntu.com/community/UFW`. Last accessed 26th of April 2016.

[UNA] Unattended Updates Ubuntu Documentation. `https://help.ubuntu.com/12.04/serverguide/automatic-updates.html`. Last accessed 26th of April 2016.

[UNI] Uninett website. `https://www.uninett.no/`. Last accessed 8th of June 2016.

[UVA] UVa Online Judge. `https://uva.onlinejudge.org/`. Last accessed 2nd of May 2016.

[VIM] Vim website. `http://www.vim.org/`. Last accessed 9th of May 2016.

[VIR] Python Virtual Environment - virtualenv. `https://virtualenv.pypa.io/en/latest/`. Last accessed 20th of April 2016.

[WER] Werkzeug Utility Library. `http://werkzeug.pocoo.org/`. Last accessed 25th of April 2016.

[WMMY93] Ronald E Walpole, Raymond H Myers, Sharon L Myers, and Keying Ye. *Probability and statistics for engineers and scientists*, volume 5. Macmillan New York, 1993.

[XU3a] Odroid XU3. `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127`. Last accessed 4th of April 2016.

[XU3b] Odroid XU3 Board Block Diagram. `http://dn.odroid.com/homebackup/201407071202252748.jpg`. Last accessed 14th of April 2016.

Appendix A

User Test Material

A.1 Digital Appendix

The raw data used for statistical analysis in Gnumeric [GNU] is included in the digital appendix. The files *Climbing_Mont_Blanc.xlsx* and *TDT4200_Parallel_Computing.xlsx* contains the survey responses of the user test conducted in this thesis and the user test conducted during the specialization project repsectively. The file *TDT4900MasterResults.xlsx* contains the results of the F-tests, T-tests, and Wilcoxon-Mann-Whitney tests.

A.2 User Study Questionnaire

What year of study are you in?

- 1st year
- 2nd year
- 3rd year
- 4th year
- 5th year
- Other: specify

Which of the following best describes your study programme?

- Computer Science at IDI
- Computer Science but not at IDI
- Electronics at NTNU
- Cybernetics at NTNU

- Mathematics or Physics at NTNU
- PhD student
- Other: specify

How would you rate the Climbing Mont Blanc system in general?

- Very poor
- Poor
- Neutral
- Good
- Very good

It was easy to use the system.

- Strongly disagree
- Disagree
- Neither agree or disagree
- Agree
- Strongly agree

How would you rate the usability of the Climbing Mont Blanc system?

- Very poor
- Poor
- Neutral
- Good
- Very good

Was it difficult to learn how to use the Climbing Mont Blanc system?

- Very hard
- Hard
- Neutral
- Easy
- Very easy

Are there any tasks or actions that you feel cumbersome or hard to perform? Please specify if any.
Textual based

How would you rate the design of the Climbing Mont Blanc user interface?

- Very poor
- Poor
- Neutral
- Good
- Very good

Are there any parts of the design that feel redundant, unlogical or confusing in any way? Please specify
Textual based

How would you rate the process of uploading and running a program?

- Very hard
- Hard
- Neutral
- Easy
- Very easy

How satisfied are you with the feedback given by the Climbing Mont Blanc system?

- Not satisfied at all
- A little bit satisfied
- Neutral
- Satisfied
- Very satisfied

How would you rate the information on the HowTo-page?

- Strongly disagree
- Disagree
- Neither agree or disagree
- Agree
- Strongly agree

Any information that is missing on the HowTo-page? Please specify.
Textual based

Have you discovered any bugs? If you have, please try to describe these.
Textual based

Any other comments on the Climbing Mont Blanc system usability or the system in general? Are there any missing features?
Textual based

A.3 User Experiment Tasks

Climbing Mont Blanc Tasks

All tasks are to be done on the Climbing Mont Blanc system, reached at http://climb-dev.idi.ntnu.no. The tasks should be done in order. If you have questions about the system, they will be answered if they are not related to the tasks. You can abort the test at any time.

Task 1:
Create a user and log into the system.

Task 2:
Join the group "Usability Test".

Task 3:
Solve one of the problems:

1. Hello World
2. Reverse String
3. Prime Numbers
4. WERTYU

Submit and run when you think you got a valid solution. Do not route away from the problem-view while the program executes.

When you get an accepted run, re-submit the same files two more times and run the submissions.

Task 4:
Find the submissions with the fastest execution time, lowest energy usage, and lowest Energy Delay Product (EDP) in the Public Highscore list.

Task 5:
Find which of your own submissions that have the fastest execution time, lowest energy usage, and lowest Energy Delay Product (EDP) in the Public Highscore list.

Task 6:
Re-run one of your past submissions on the problem you chose in Task 3.

Task 7:
Repeat Task 4 for the "Usability Test"-group highscore list.

Task 8:
Repeat Task 5 for the “Usability Test”-group highscore list.

Task 9:
Create your own group. Name it <Your username>’s Group.

Task 10:
Add the “Hello World”-problem to your group. Also, add the user sindrma to your group.

Task 11:
Check out your groups state on the “Hello World”-problem.
Check out sindrma’s group state on all the problems in your group.

Task 12 (if time permits):
Change your users password and/or email.

Task 13:
Fill out this survey:
https://docs.google.com/forms/d/1rmrWo41UCArbhN_GpTPpKqT9OQAr5LKn3uGvvZZedz8/viewform

Thank you for your participation!

A.4 User Test Results

Are there any tasks or actions that you feel cumbersome or hard to perform? Please specify if any.

- Unclear whether to zip folders, files etc. Should instead be a small label next to the upload button, instead of link to the HowTo page. Upload should accept single or multi source file upload. Might be easier to upload from Unix systems (*Count: 9*).
- Rerunning code is implicitly discouraged compared, and users should be notified about the behaviour (*Count: 1*).
- Waiting a long time in the run-queue (*Count: 2*).

Are there any parts of the design that feel redundant, unlogical or confusing in any way? Please specify.

- Went into group usability test and could not find the public high-score list, needed to go to start page to find it (*Count: 1*, the user were probably not logged into the system).
- The x on the “Show error”- button is making the users think that it removes the error message (*Count: 1*).
- Flickering in Highscore-list (*Count: 1*).
- The role of groups in the system is unclear (*Count: 1*).
- No need for both email address and user name in the system, as it is hard to remember both (*Count: 1*).
- Weird to add people to a group without their consent (*Count: 1*).
- Double arrow on on table headers feels confusing, as it can be interpreted as the list can be sorted in both ascending and descending order. However, the list can only be sorted in ascending order (*Count: 1*).
- The whole UI feels a little disorganized and tables are not aligned. Buttons appear without a button row. The dropdown menu + button above the high score table is not intuitive. It changes title, and its hard to understand how “Public” is different from the group score (*Count: 1*).

Do you feel any feedback is missing in the Climbing Mont Blanc system? Please specify.

- Feedback is good and the system clarifies the problem if there is errors (*Count: 1*).
- Up-to-date run queue status, such as position in queue and expected queue time. Updated run time information (*Count: 11*).

- Little feedback during heavy system load, and show estimated progress bar instead of spinner (*Count: 2*).
- When adding problems to your group, the "add problem"-button should be "grayed-out" until a valid problem name is specified. Currently, no error message is shown when giving a invalid problem name and trying to add the problem. The same applies to the "add member"-button on the same page (*Count: 1*).
- "Runtime error in small input!"-message does not give sense to untrained users (*Count: 1*).

Any information that is missing on the HowTo-page? Please specify.

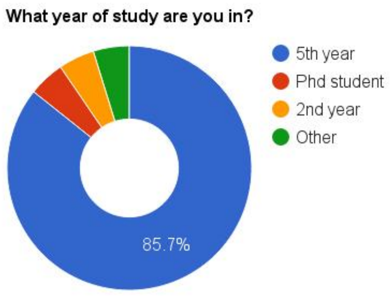
- Contains to much text (*Count: 1*).
- More information about run procedure, queuing multiple submissions, and possible output (*Count: 1*).
- Exact information about format and file name conventions in uploads, as well as what files to include (*Count: 2*).
- Broken links (*Count: 2*, should be ignored as it had to do with Nginx problems during user test).
- Should specify C++ version and C io examples (*Count: 1*).

Have you discovered any bugs? If you have, please try to describe these.

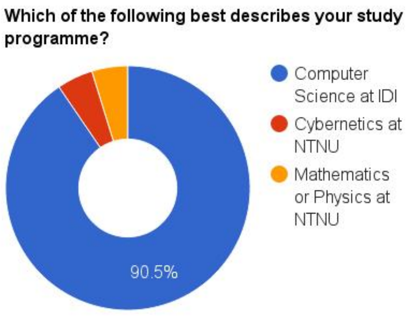
- Password and email update fields indicating forgotten input even after updating correctly. The message should be removed in this situation (*Count:*).
- Broken links (*Count: 3*, should be ignored as it had to do with Nginx problems during user test).
- Flickering in Highscore-list (*Count: 2*).
- Highscore list, seemed to refresh some values too slow, new results took a little time to view (*Count: 1*).
- Safari seemed to just show upload toast/annotation and not actually upload the file: seemed as the upload process did not start when button was pressed (*Count: 3*).
- Constraints should be specified in problem descriptions (*Count: 2*).

Any other comments on the Climbing Mont Blanc system usability or the system in general? Are there any missing features?

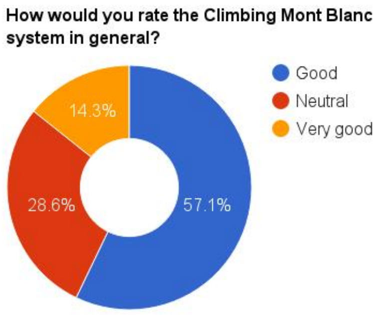
- Great system and fine usability. Cool that the system measures energy efficiency. Clear and well represented results. (*Count: 1*).
- The user interface is stellar. It could be useful to include information on the expected precision and noise levels of time and energy measurements - or ideally, if the system submission speed is improved then reporting averages and variance from multiple runs of your program (*Count: 1*).
- Browser IDE? would be A LOT of work, but nice - another prerequisite would also be that the server would let the user return to the IDE while waiting for the error message, including a quicker response time (*Count: 1*).
- Not possible to upload files using Safari (*Count: 1*).
- Information about placement in queue and other run queue information when the queue contains a lot of submissions. The time it takes to get result after hitting run is to long (*Count: 5*).
- The possibility to zip and upload several files in one go (*Count: 1*).



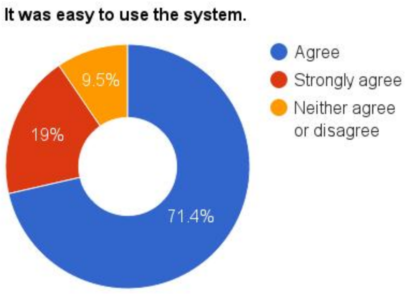
(a) *



(b) *

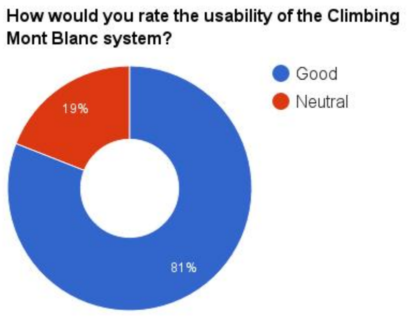


(c) A1

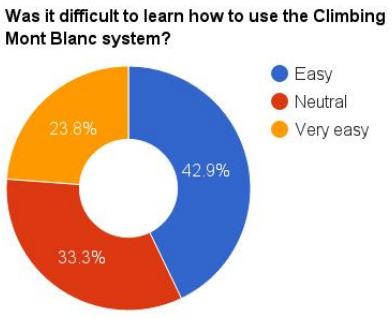


(d) B1

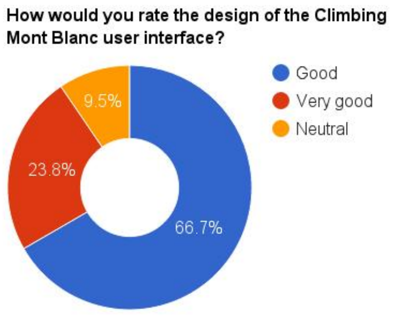
Figure A.1: Multiple Choice Results



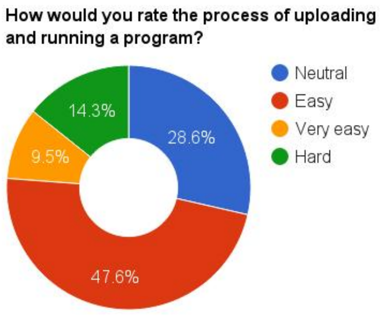
(a) A2



(b) C1

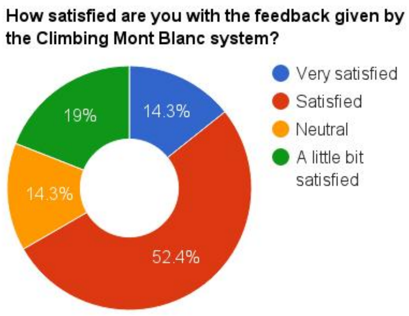


(c) A3

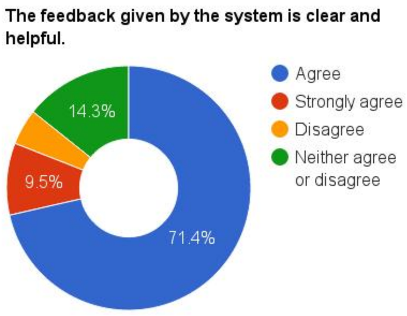


(d) C2

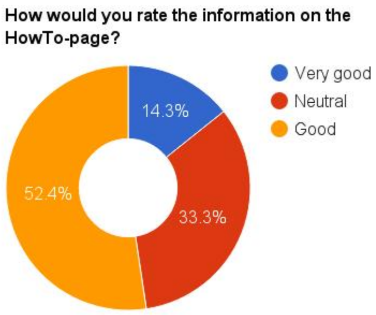
Figure A.2: Multiple Choice Results (continuation of Figure A.1)



(a) D1



(b) B2



(c) A4

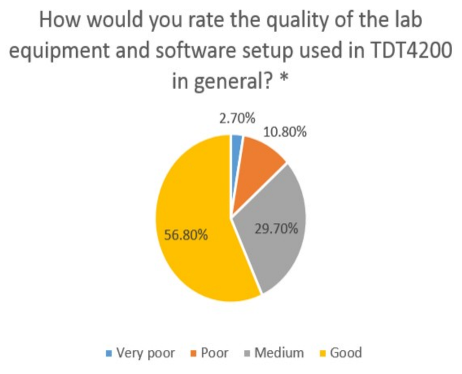
Figure A.3: Multiple Choice Results (continuation of Figure A.2)

A.5 TDT4200 User Study Results

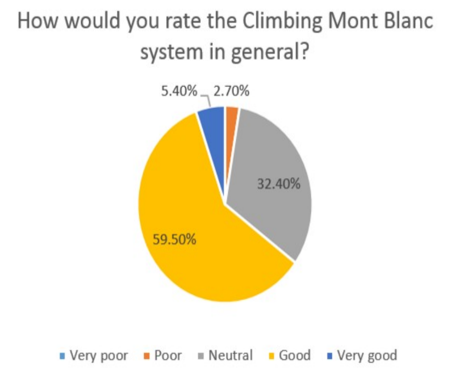
There were in total 37 students completing the survey. Figures A.4 and A.5 shows the results of the multiple choice questions related to CMB. Some background information about the participants is also presented in Figure A.6. The Figure describes the distribution amongst the year of study and area of study. A summary of the feedback given in the textual based questions and feedback received in class is found below:

- Feedback given from the CMB system should be improved. Both regarding compilation errors and runtime errors.
- The format and how to structure the zip to be uploaded was unclear. Would be nice to be able to upload single source files.
- Submission of zip files from OSX did not work.
- The delivery zip format through ItsLearning and CMB varied, which further lead to some confusion.
- CMB had days with long run queue time.
- Compilation and running of code should be done in one action.
- One should be able to delete failed submissions as a user.
- Running the same submission one more time should result in a new submission to the high score list, not an updated entry.
- Sometimes, submissions that were chosen to be private were shown. The bug becomes present when sorting the highscore list on one of the other metrics.
- The login procedure requires too many clicks.
- CMB should support command line interface instead of the User Interface to compile and run programs.
- Submissions is not visible accessing the problem page from a group view, only when accessing the problem from the public list of problems.
- Expected compilation time and running time should be added, and also dynamic update the highscore list when runs have completed.
- The submissions should display the lines of code and more information about the uploaded files.
- The sorting of the highscore list should consider energy as a secondary priority next to running time when sorting the highscore list.
- More languages should be supported.
- Users should be able to specify compiler flags.

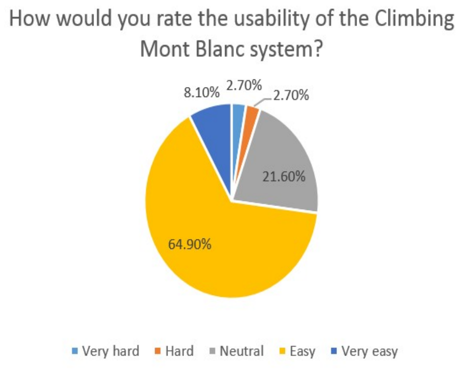
- Detect content within the uploaded zip file.
- CMB is a really good idea; you should try to improve it.
- It needs some improvement, but, all in all, it is a good system.
- Overall usability is good. But the system might need some overhaul of its components.



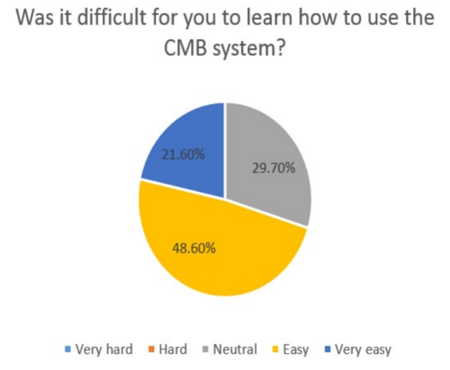
(a)



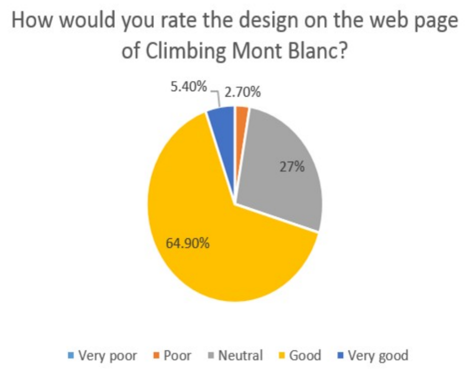
(b)



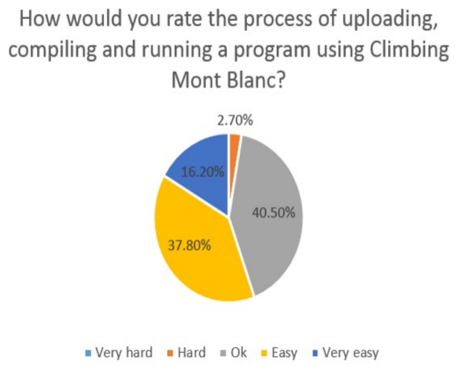
(c)



(d)

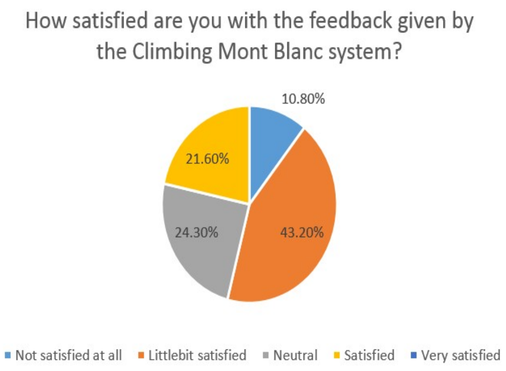


(e)

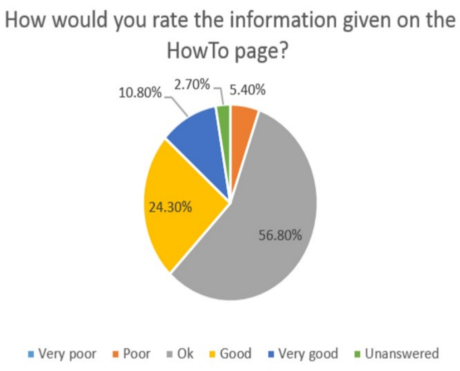


(f)

Figure A.4: CMB Related Multiple Choice Results

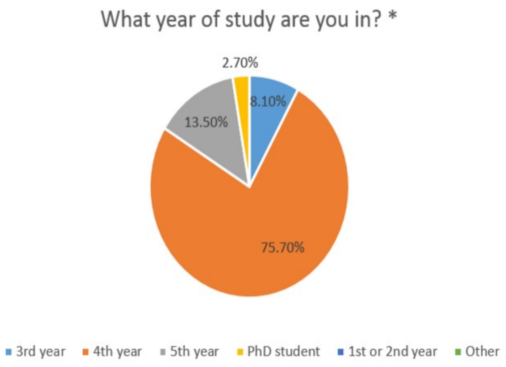


(a)

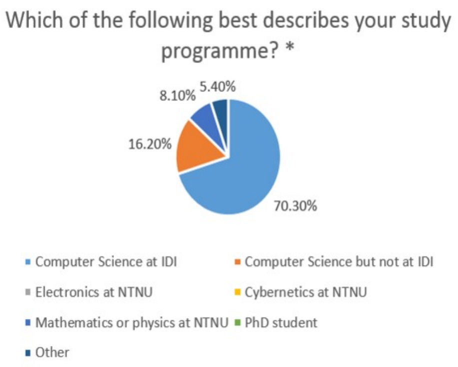


(b)

Figure A.5: CMB Related Multiple Choice Results (continuation of A.4)



(a)



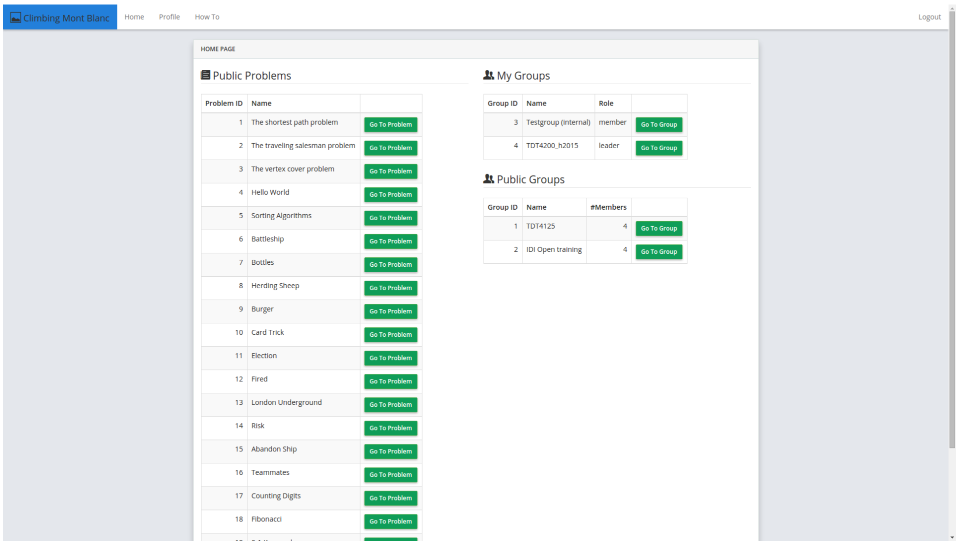
(b)

Figure A.6: Participant Distribution

Appendix B

System Frontend Screenshots

In this chapter, some screenshots from system frontend views of CMB version one and two are presented for easy comparison of a subset of the views present in the two versions.



HOME PAGE

Public Problems

Problem ID	Name	
1	The shortest path problem	Go To Problem
2	The traveling salesman problem	Go To Problem
3	The vertex cover problem	Go To Problem
4	Hello World	Go To Problem
5	Sorting Algorithms	Go To Problem
6	Battleship	Go To Problem
7	Bottles	Go To Problem
8	Herding Sheep	Go To Problem
9	Burger	Go To Problem
10	Card Trick	Go To Problem
11	Election	Go To Problem
12	Fired	Go To Problem
13	London Underground	Go To Problem
14	Risk	Go To Problem
15	Abandon Ship	Go To Problem
16	Teammates	Go To Problem
17	Counting Digits	Go To Problem
18	Fibonacci	Go To Problem

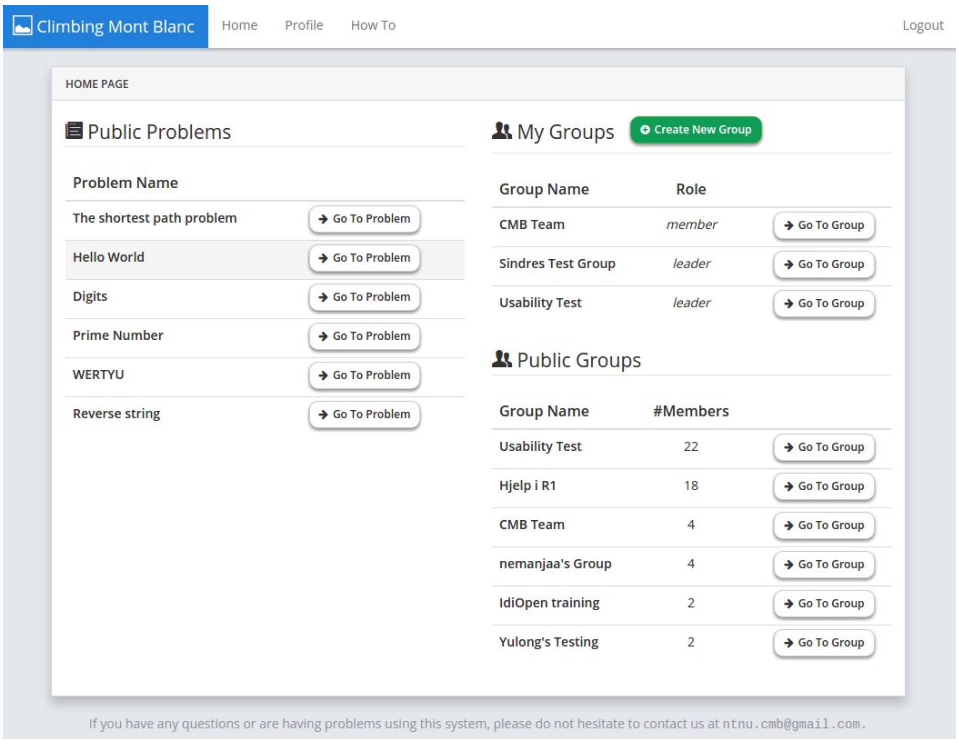
My Groups

Group ID	Name	Role	
3	Testgroup (internal)	member	Go To Group
4	TDT4200_h2015	leader	Go To Group

Public Groups

Group ID	Name	#Members	
1	TDT4125	4	Go To Group
2	IDI Open training	4	Go To Group

Figure B.1: The old home-view.



HOME PAGE

Public Problems

Problem Name	
The shortest path problem	Go To Problem
Hello World	Go To Problem
Digits	Go To Problem
Prime Number	Go To Problem
WERTYU	Go To Problem
Reverse string	Go To Problem

My Groups

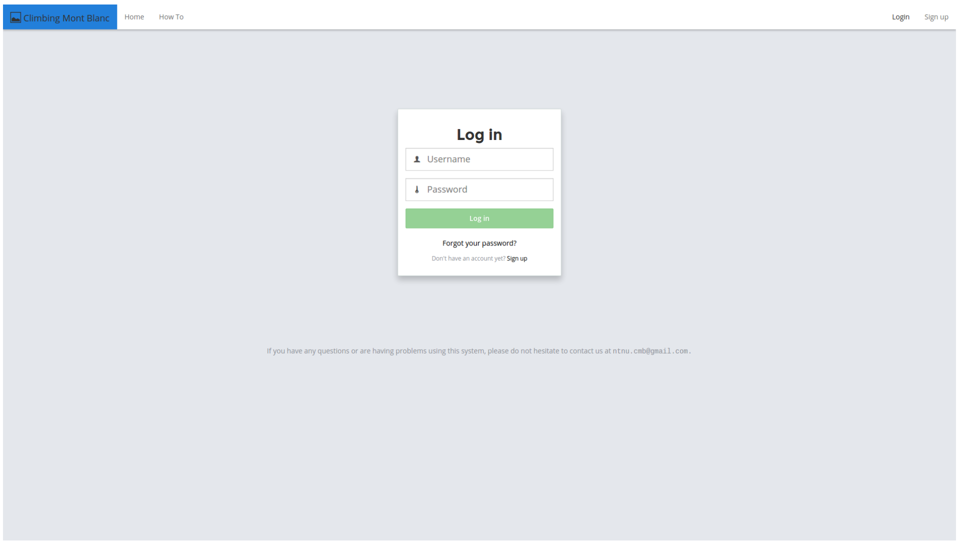
Group Name	Role	
CMB Team	*member*	Go To Group
Sindres Test Group	*leader*	Go To Group
Usability Test	*leader*	Go To Group

Public Groups

Group Name	#Members	
Usability Test	22	Go To Group
Hjelp i R1	18	Go To Group
CMB Team	4	Go To Group
nemanjaa's Group	4	Go To Group
IdiOpen training	2	Go To Group
Yulong's Testing	2	Go To Group

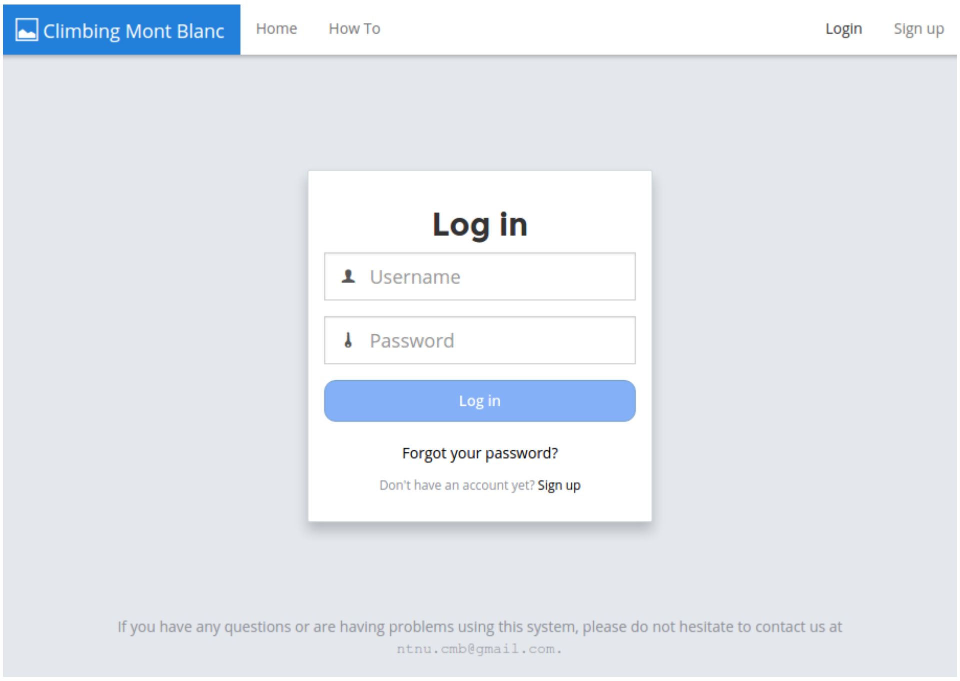
If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.2: The new home-view.



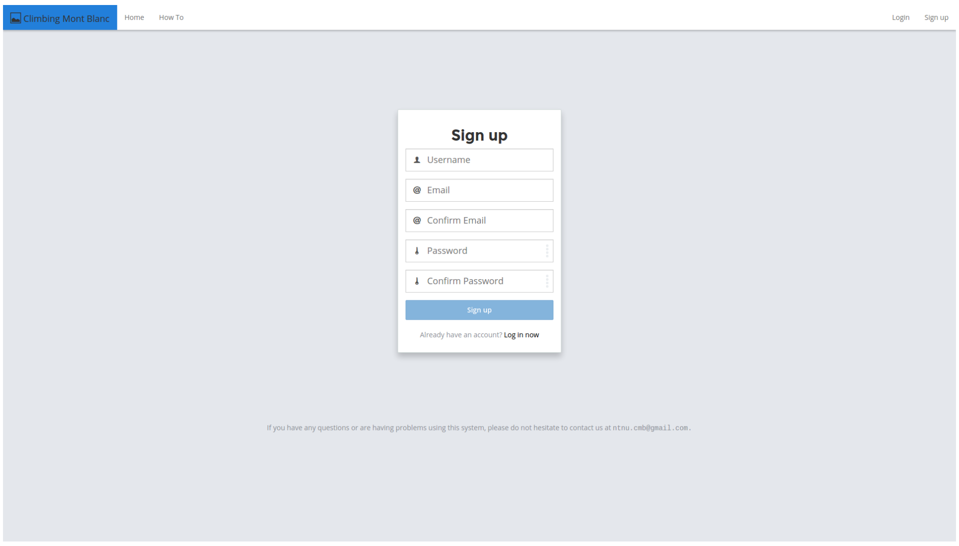
If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.3: The old login-view.



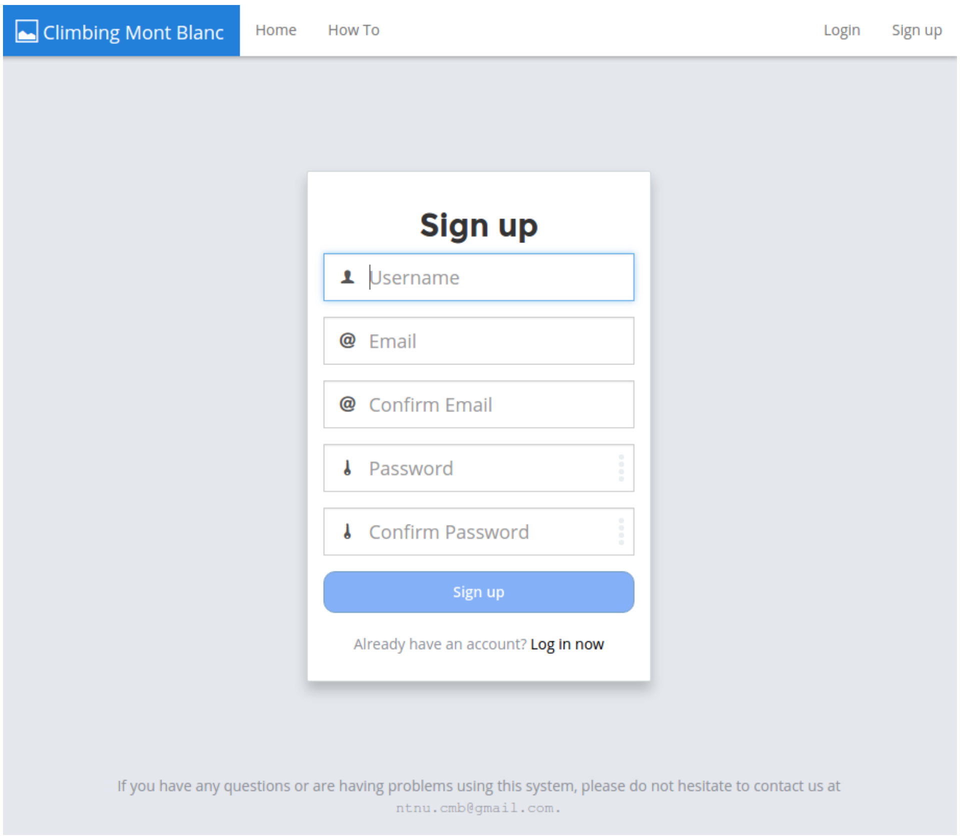
If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.4: The new login-view.



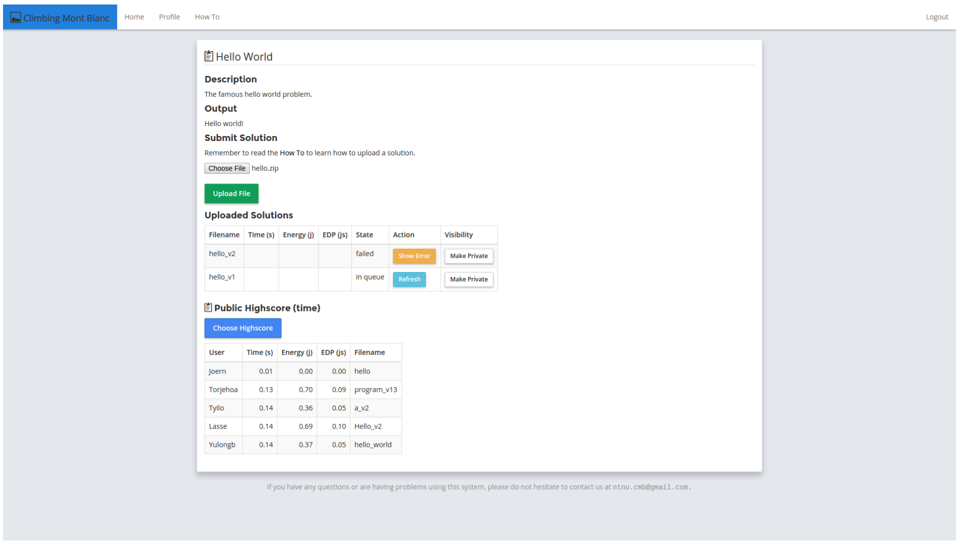
If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.5: The old signup-view.



If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

Figure B.6: The new signup-view.



Hello World

Description

The famous hello world problem.

Output

Hello world!

Submit Solution

Remember to read the How To to learn how to upload a solution.

Uploaded Solutions

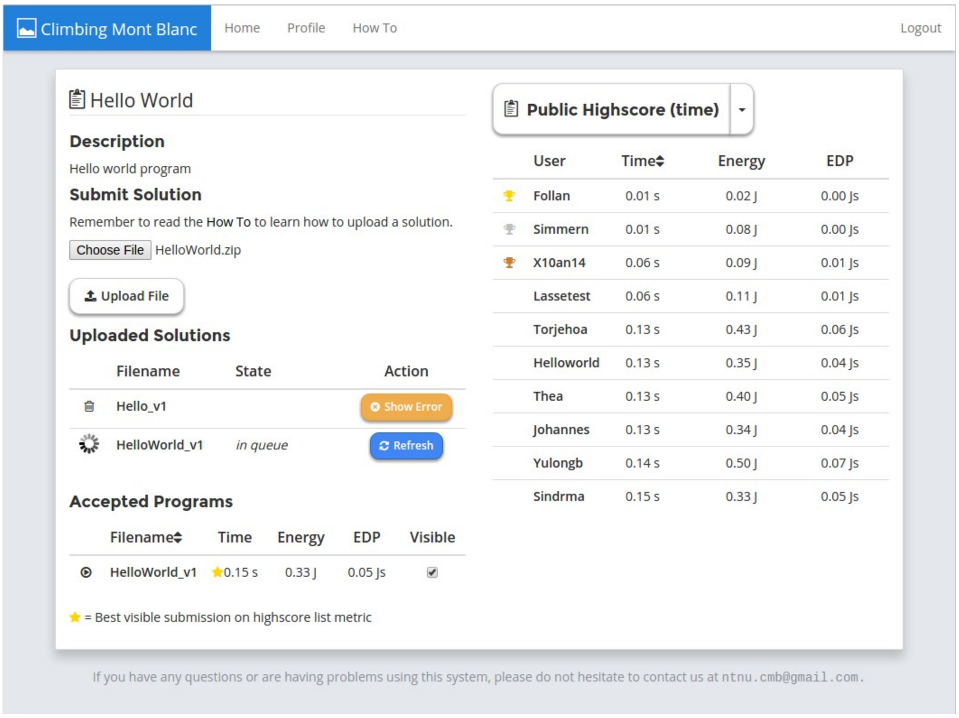
Filename	Time (s)	Energy (j)	EDP (js)	State	Action	Visibility
hello_v2				failed	Show Error	Make Private
hello_v1				in queue	Refresh	Make Private

Public Highscore (time)

User	Time (s)	Energy (j)	EDP (js)	Filename
Joern	0.01	0.00	0.00	hello
Torjehoa	0.13	0.70	0.09	program_v13
Tyilo	0.14	0.36	0.05	a_v2
Lasse	0.14	0.69	0.10	Hello_v2
Yulongb	0.14	0.37	0.05	hello_world

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.7: The old problem-view.



Hello World

Description

Hello world program

Submit Solution

Remember to read the How To to learn how to upload a solution.

Uploaded Solutions

	Filename	State	Action
	Hello_v1		Show Error
	HelloWorld_v1	*in queue*	Refresh

Accepted Programs

	Filename	Time	Energy	EDP	Visible
	HelloWorld_v1	★0.15 s	0.33 J	0.05 Js	✓

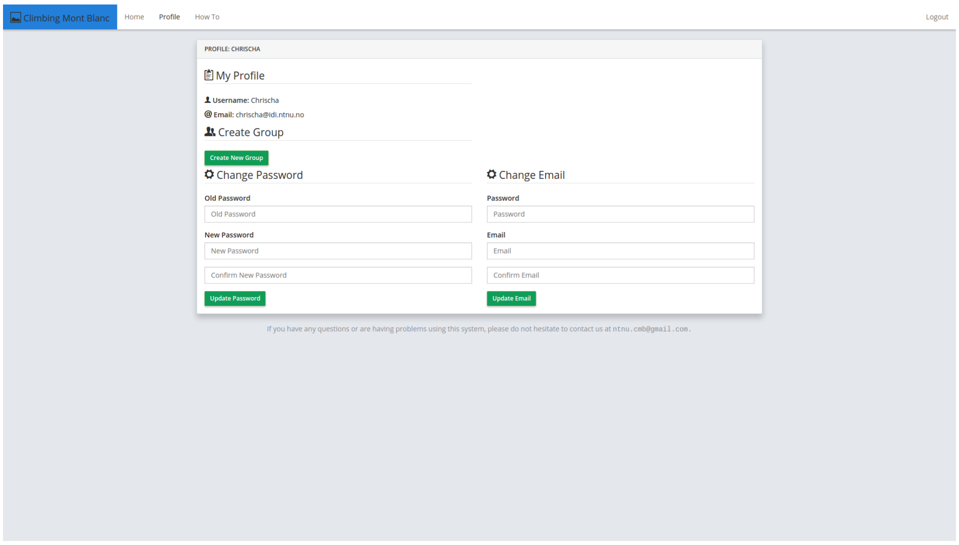
★ = Best visible submission on highscore list metric

Public Highscore (time)

	User	Time	Energy	EDP
🏆	Follan	0.01 s	0.02 J	0.00 Js
🏆	Simmern	0.01 s	0.08 J	0.00 Js
🏆	X10an14	0.06 s	0.09 J	0.01 Js
	Lassetest	0.06 s	0.11 J	0.01 Js
	Torjehoa	0.13 s	0.43 J	0.06 Js
	Helloworld	0.13 s	0.35 J	0.04 Js
	Thea	0.13 s	0.40 J	0.05 Js
	Johannes	0.13 s	0.34 J	0.04 Js
	Yulongb	0.14 s	0.50 J	0.07 Js
	Sindrma	0.15 s	0.33 J	0.05 Js

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.8: The new problem-view.



PROFILE: CHRISCHA

My Profile

Username: Chrischa
Email: chrischa@idi.ntnu.no

Create Group

Change Password

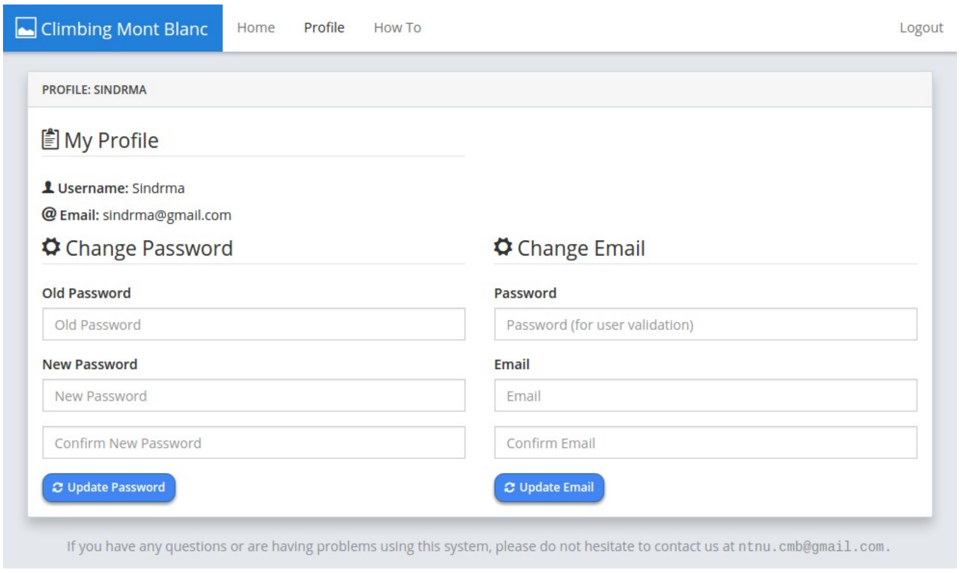
Old Password: Old Password
New Password: New Password
Confirm New Password

Change Email

Password: Password
Email: Email
Confirm Email

If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

Figure B.9: The old profile-view.



PROFILE: SINDRMA

My Profile

Username: Sindrma
Email: sindrma@gmail.com

Change Password

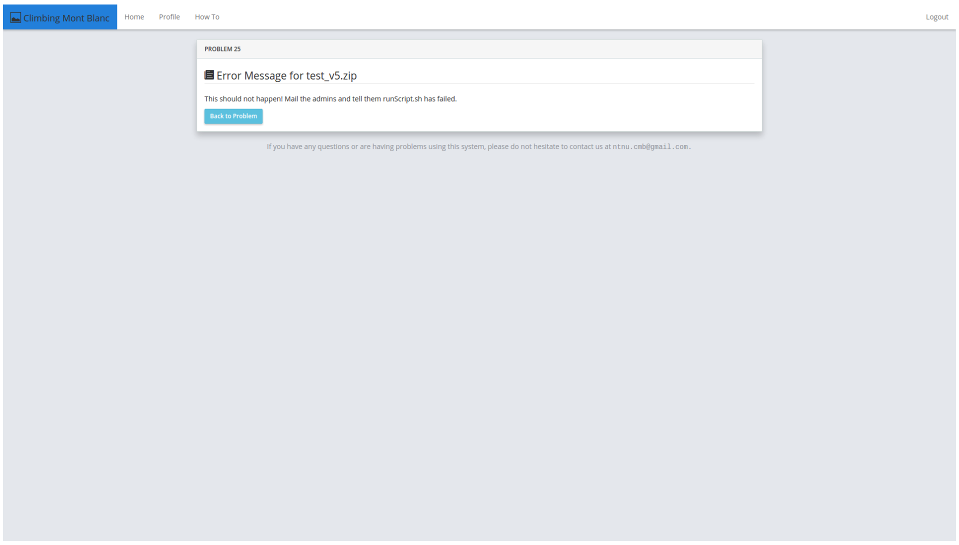
Old Password: Old Password
New Password: New Password
Confirm New Password

Change Email

Password: Password (for user validation)
Email: Email
Confirm Email

If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

Figure B.10: The new profile-view.



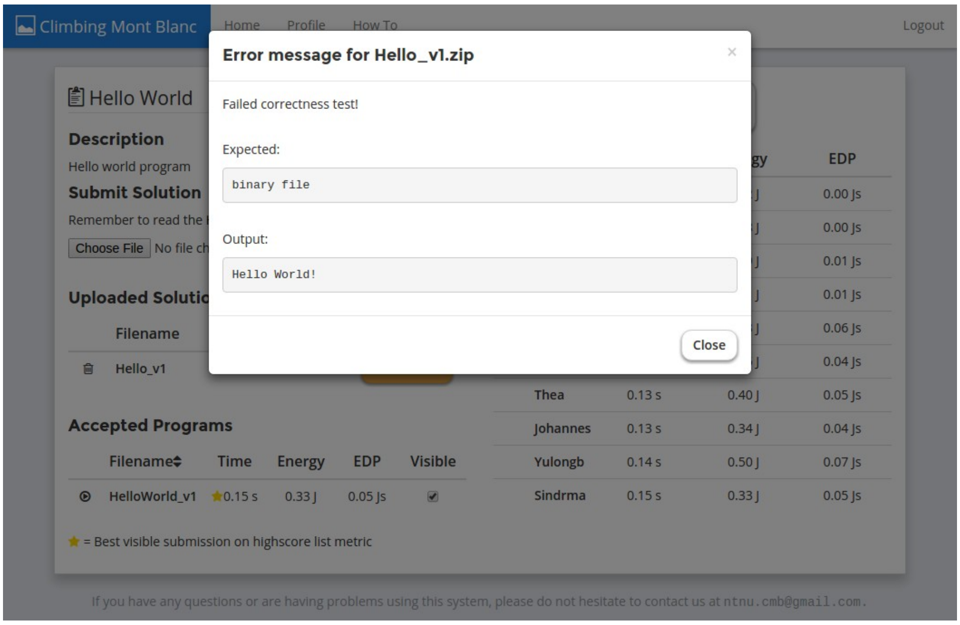
PROBLEM 25

Error Message for test_v5.zip

This should not happen! Mail the admins and tell them runScript.sh has failed.

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.11: The old error-message-view.



Error message for Hello_v1.zip

Failed correctness test!

Expected:

```
binary file
```

Output:

```
Hello World!
```

Hello World

Description

Hello world program

Submit Solution

Remember to read the

Uploaded Solutio

	Filename
🗑	Hello_v1

Accepted Programs

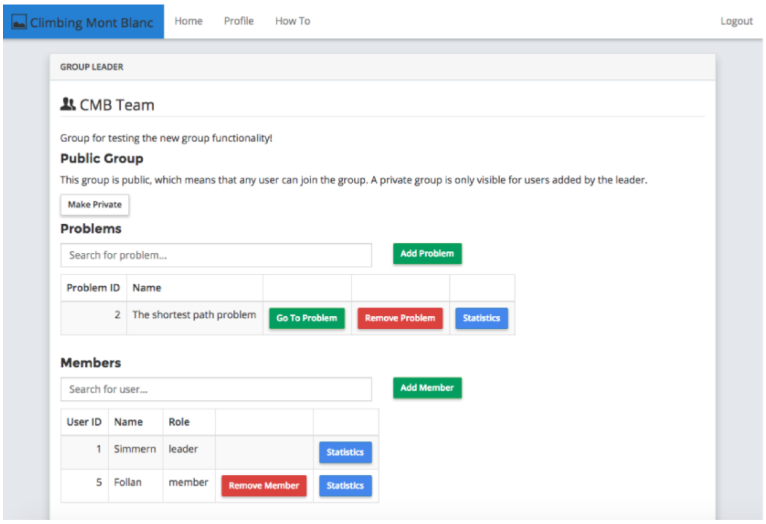
	Filename	Time	Energy	EDP	Visible
⊙	HelloWorld_v1	★0.15 s	0.33 J	0.05 Js	✓

★ = Best visible submission on highscore list metric

		gy	EDP
		J	0.00 Js
		J	0.00 Js
		J	0.01 Js
		J	0.01 Js
		J	0.06 Js
		J	0.04 Js
Thea	0.13 s	0.40 J	0.05 Js
Johannes	0.13 s	0.34 J	0.04 Js
Yulongb	0.14 s	0.50 J	0.07 Js
Sindrma	0.15 s	0.33 J	0.05 Js

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.12: The new error modal.



GROUP LEADER

CMB Team

Group for testing the new group functionality!

Public Group

This group is public, which means that any user can join the group. A private group is only visible for users added by the leader.

Make Private

Problems

Search for problem... | Add Problem

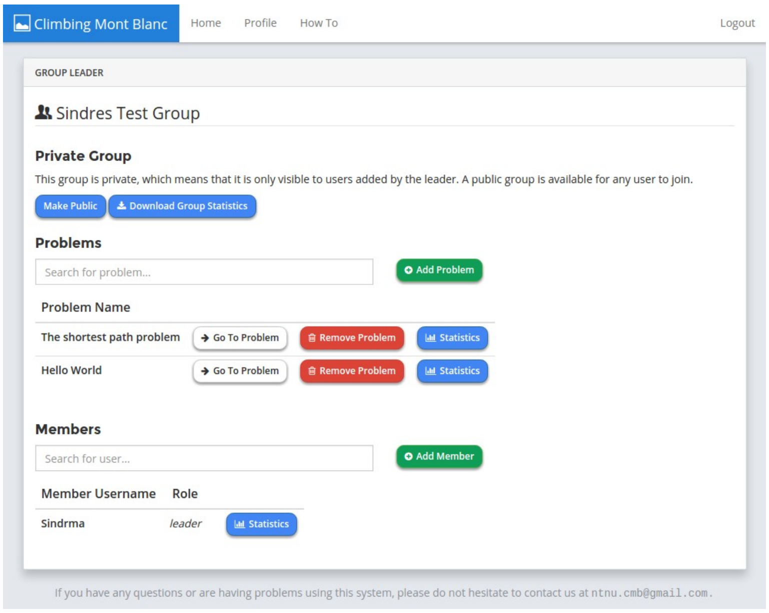
Problem ID	Name			
2	The shortest path problem	Go To Problem	Remove Problem	Statistics

Members

Search for user... | Add Member

User ID	Name	Role		
1	Simmern	leader		Statistics
5	Follan	member	Remove Member	Statistics

Figure B.13: The old group leader-view (taken from Follan and Støa [FS15]).



GROUP LEADER

Sindres Test Group

Private Group

This group is private, which means that it is only visible to users added by the leader. A public group is available for any user to join.

Make Public | Download Group Statistics

Problems

Search for problem... | Add Problem

Problem Name			
The shortest path problem	Go To Problem	Remove Problem	Statistics
Hello World	Go To Problem	Remove Problem	Statistics

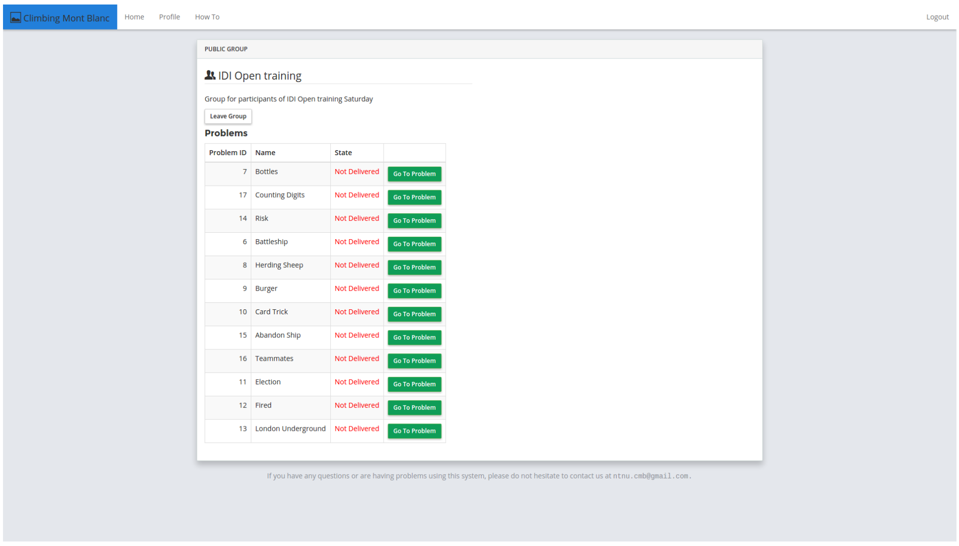
Members

Search for user... | Add Member

Member Username	Role	
Sindrma	*leader*	Statistics

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.14: The new group leader-view.



PUBLIC GROUP

IDI Open training

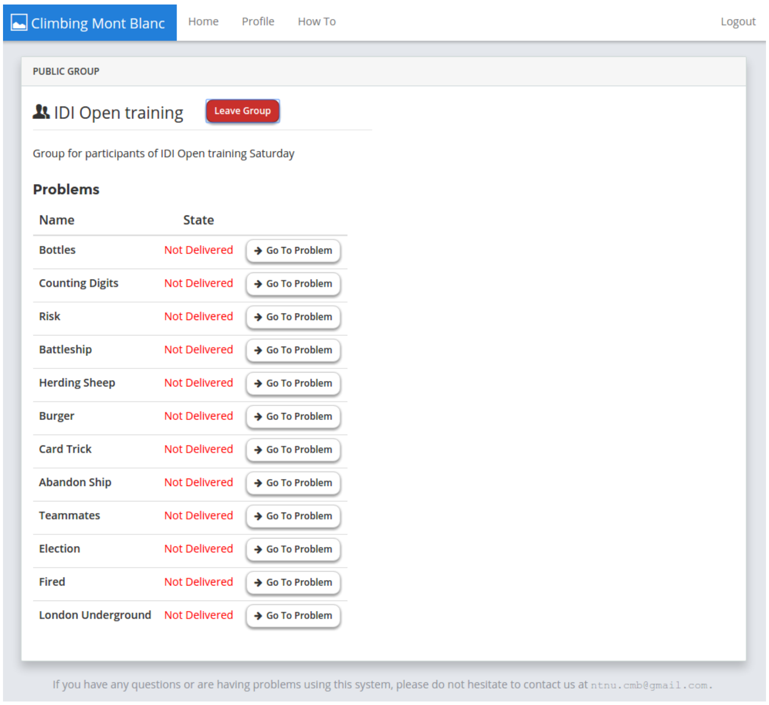
Group for participants of IDI Open training Saturday

Problems

Problem ID	Name	State	
7	Bottles	Not Delivered	Go To Problem
17	Counting Digits	Not Delivered	Go To Problem
14	Risk	Not Delivered	Go To Problem
6	Battleship	Not Delivered	Go To Problem
8	Herding Sheep	Not Delivered	Go To Problem
9	Burger	Not Delivered	Go To Problem
10	Card Trick	Not Delivered	Go To Problem
15	Abandon Ship	Not Delivered	Go To Problem
16	Teammates	Not Delivered	Go To Problem
11	Election	Not Delivered	Go To Problem
12	Fired	Not Delivered	Go To Problem
13	London Underground	Not Delivered	Go To Problem

If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

Figure B.15: The old group-view while member.



PUBLIC GROUP

IDI Open training

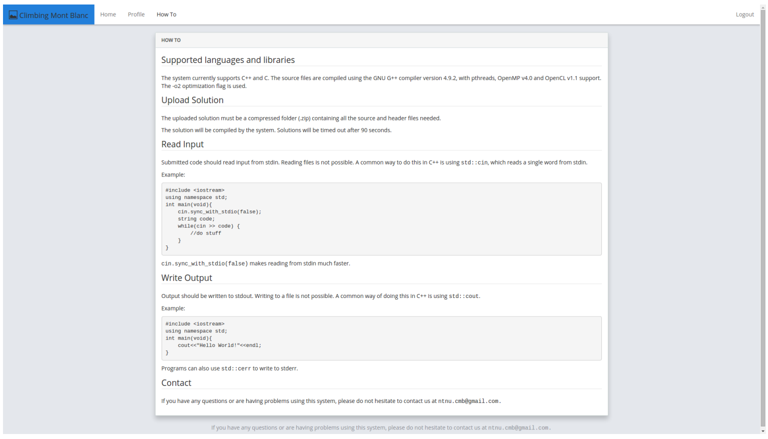
Group for participants of IDI Open training Saturday

Problems

Name	State	
Bottles	Not Delivered	Go To Problem
Counting Digits	Not Delivered	Go To Problem
Risk	Not Delivered	Go To Problem
Battleship	Not Delivered	Go To Problem
Herding Sheep	Not Delivered	Go To Problem
Burger	Not Delivered	Go To Problem
Card Trick	Not Delivered	Go To Problem
Abandon Ship	Not Delivered	Go To Problem
Teammates	Not Delivered	Go To Problem
Election	Not Delivered	Go To Problem
Fired	Not Delivered	Go To Problem
London Underground	Not Delivered	Go To Problem

If you have any questions or are having problems using this system, please do not hesitate to contact us at ntnu.cmb@gmail.com.

Figure B.16: The new group-view while member.



HOW TO

Supported languages and libraries

The system currently supports C++ and C. The source files are compiled using the GNU G++ compiler version 4.9.2, with pthreads, OpenMP v4.0 and OpenCL v1.1 support. The -o2 optimization flag is used.

Upload Solution

The uploaded solution must be a compressed folder (.zip) containing all the source and header files needed.

The solution will be compiled by the system. Solutions will be timed out after 90 seconds.

Read Input

Submitted code should read input from stdin. Reading files is not possible. A common way to do this in C++ is using `std::cin`, which reads a single word from stdin.

Example:

```
#include <iostream>
using namespace std;
int main(void){
    cin.sync_with_stdio(false);
    string code;
    while(cin >> code) {
        //do stuff
    }
}
```

`cin.sync_with_stdio(false)` makes reading from stdin much faster.

Write Output

Output should be written to stdout. Writing to a file is not possible. A common way of doing this in C++ is using `std::cout`.

Example:

```
#include <iostream>
using namespace std;
int main(void){
    cout<<"Hello World!"<<endl;
}
```

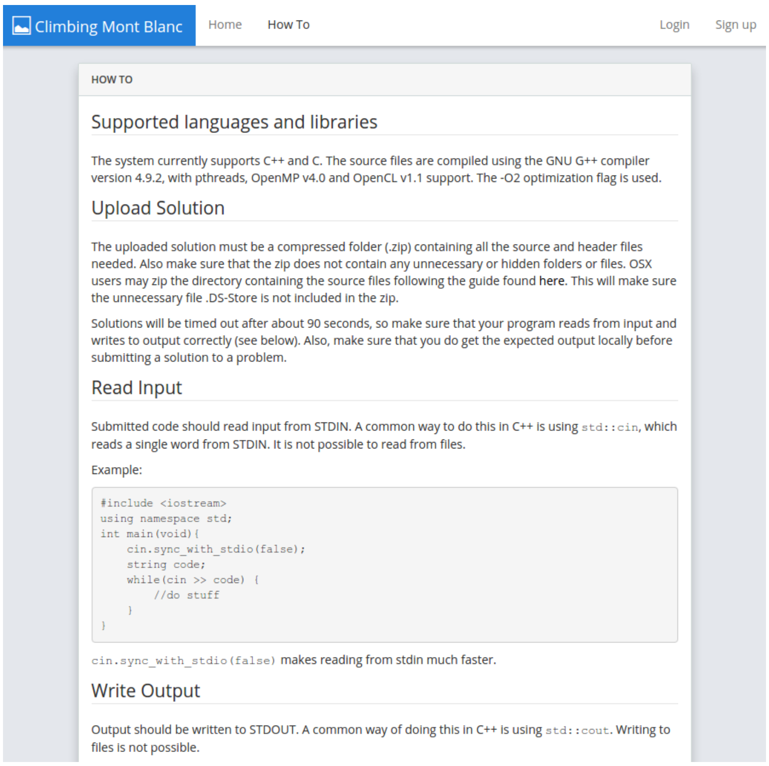
Programs can also use `std::cerr` to write to stderr.

Contact

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

If you have any questions or are having problems using this system, please do not hesitate to contact us at `ntnu.cmb@gmail.com`.

Figure B.17: The old HowTo-view.



HOW TO

Supported languages and libraries

The system currently supports C++ and C. The source files are compiled using the GNU G++ compiler version 4.9.2, with pthreads, OpenMP v4.0 and OpenCL v1.1 support. The -O2 optimization flag is used.

Upload Solution

The uploaded solution must be a compressed folder (.zip) containing all the source and header files needed. Also make sure that the zip does not contain any unnecessary or hidden folders or files. OSX users may zip the directory containing the source files following the guide found **here**. This will make sure the unnecessary file .DS-Store is not included in the zip.

Solutions will be timed out after about 90 seconds, so make sure that your program reads from input and writes to output correctly (see below). Also, make sure that you do get the expected output locally before submitting a solution to a problem.

Read Input

Submitted code should read input from STDIN. A common way to do this in C++ is using `std::cin`, which reads a single word from STDIN. It is not possible to read from files.

Example:

```
#include <iostream>
using namespace std;
int main(void){
    cin.sync_with_stdio(false);
    string code;
    while(cin >> code) {
        //do stuff
    }
}
```

`cin.sync_with_stdio(false)` makes reading from stdin much faster.

Write Output

Output should be written to STDOUT. A common way of doing this in C++ is using `std::cout`. Writing to files is not possible.

Figure B.18: The new HowTo-view.

Appendix C

Backlog

- **Bugs and known issues**:
  - Remove flickering in the highscore list during Socket.IO updates: The frontend fetches to much information when receiving events from the server.
  - Should add cascading deletes on Problem-table: To remove dangling submissions if problems are to be deleted in the future.
- **Stability**:
  - Improve stability of measurements: Extend the backend with the possibility to assign programs to cores at the backend. Better control over processor and board temperature is also wanted.
  - Improved stability test: The stability test should automatically calculate mean, standard deviation and relative standard deviation of runs.
  - Automatic system monitoring and recovery: The system should be able to detect irregularities such as long submission queues, and take action automatically if things should fail.
  - Automatic file checks on server: Should check the format and content of submitted files.
  - Port bash scripts to Python code: Will make it easier to write unit tests, and hopefully make the system more robust and stable.
  - Compile threads and cross-compiler: Should be introduced if applicable to restrict inline assembly code and make the compile process more stable on both the server and backend.
- **Scalability**
  - Broker: Extend with multiple Odroid XU3 boards. If there is a submission in the queue, the broker should assign a board to the submission for execution.

  - Multiple architectures: Extend with different architectures, like Odroid XU4 and others.

- **Usability**:
  - Improved feedback: Should report placement of submissions in the run queue dynamically. Should also report expected running time with progress bars, and the server should kill submissions which timeouts.
  - Upload improvement: Should be possible to upload single source and header files. Also checking of the zip file content, and automatic removal of unnecessary files is wanted.
  - Highscore list improvement: Should sort on EDP or energy next if the running time between two submission matches. It should also be possible to run the same submission again, and chose which run that should be visible in the highscore list.
  - Multiple test cases during checking: The checking of submissions should be split into stages, to more easily report.
  - Multi file upload: Should be possible to upload multiple source files to the system. Could also extend with an online code editor.
  - Compilation improvement: Users or admins should be able to specify compiler flags. The makefile should also be made on a per problem basis or on a per programming language basis, instead of having a huge static makefile. The system also lacks a good cross-compilers.
  - Discussion forum.
  - Login via Feide: It should be possible to login using your NTNU user.
  - Group extensions: Add deadline and ”Late” mark. Leaders should have the possibility to promote users to leaders.
  - Time limit per problem per language: If new languages are to be supported, there should be a designated limit on timeout for the new languages on each problem. Timeouts should be defined by administrators making problems.
  - Add supported languages: Java, Python and more languages.
  - More advanced low level statistics: Cache misses, cache hits, memory usage etc. on submissions.
  - Improve information on HowTo-page: Should display language version. Should also consider splitting it into an About- and HowTo-page.
  - News Bulletin and Newsletter.
  - Searching for problems, users, and groups.
  - Seasons: Highscores by seasons/timespan.
  - Command line interface for submitting, compiling and running programs. Release API documentation.

  - Secrecy in groups: Group results and user names should have the possibility to be secret.
  - Frontend statistics: Submission statistics such as mean and standard deviation for a problem per user, per group etc. Also, low-level statistics such as memory usage, cache information, noise in measurements etc. is wanted.
  - Placeholders should be present for empty tables.

- **Performance**:
  - Non-blocking database: The database access is as of now synchronous. For a performance gain if the system load is high, a non-blocking (asynchronous) MySQL adapter could be used in combination with the default MySQL SQLAlchemy adapter.

- **Admin Interface**
  - Boilerplate checkers in the admin interface: The admin interface should add boilerplate checkers to allow easy setup of new problems.

Appendix D

Administration

It should be noted that the information stated in this chapter was created by Follan and Støa [FS15]. The information is repeated in this chapter in a slightly rewritten and compressed form as it was requested by the main supervisor.

D.1 Bitbucket, Jenkins and Google Analytics

The Jenkins server is installed at the development server of CMB and can be reached at `http://climb-dev.idi.ntnu.no:9000`. The defined pipeline stages and setup information can be found there. The Google Analytics account can be reached at `https://www.google.com/analytics`. The git repositories are accessible via Bitbucket at `https://bitbucket.org`. Access to the services are granted by the CMB team.

D.2 Problem Descriptions Best Practices

It is important that problem descriptions are clear and unambiguous, and that all necessary information is stated in the description so that users clearly understand the problem. It is recommended to follow the following format when creating problem descriptions:

1. **Description**: An overall description of the problem.
2. **Input**: Should present the format of the input. It is also common practice to state the number of inputs as the first line in the input.
3. **Output**: Should present the format of the output.
4. **Example**: Provide an example of inputs and expected outputs. This is valuable, as users can test the example locally before submitting to the system.

Remember that good problem descriptions attracts more users and submissions.

D.3 Adding and Hiding Problems

Login as an admin user either at `http://climb-dev.idi.ntnu.no` eller `http://climb.idi.ntnu.no` depending on where the problem are to be added. Access to the admin interface can be requested by contacting the CMB team. When logged in, follow the steps below to add the problem:

1. Select "Problem" in the the topmost main menu.
2. Click "Create" in the sub menu and execute the following steps:
   - Insert problem name in the "Name" field. Do not use '/' in the problem name!
   - Add an explanation in the "Description" field. This is an HTML field.
   - Insert a date in the "Created"-field.
   - Write the name of the quantity to be optimized in the "Goodness Name" field, or leave it empty if not appropriate. Only added if needed by the checker-program (see Section D.5).
   - Note! Do not tick off the "Visible" check-box yet, wait until all problem data is uploaded (see below).
3. Insert the new problem by clicking the "Submit" button.
4. Select "Uploads" in the top level horizontal menu bar. Notice that a folder with the newly added problem name has been created, where the name is lower case and spaces are replaced with underscores.
5. Select the newly created folder and then select the "problemIO"-subfolder.
6. When in the "problemIO" subfolder, upload the following files using the "Upload File"-button. Notice there is no multi-file upload. The file names must be the following, and all files must be present.
   - *input.txt*: Input for the measured test.
   - *answer.txt*: Correct answer for the measured test.
   - *small_input.txt*: Input for the small correctness test.
   - *small_answer.txt*: Correct answer for the small correctness test.
   - *checker.cpp*: A problem checker written in C++. It will automatically be compiled to a checker executable. The checker should return 0 on success, any other number on failure.
7. This step can be skipped if the checker is correct. The checker is executed in the following way:

```
./checker input.txt output.txt answer.txt
```

where *output.txt* is the submitted program's output. This means the third command line argument is the file name of the output of a submitted program (argv[2]), and can be created by running the program with the arguments present in the *input.txt* file.

8. Return to the "Problem"-tab, then modify the problem by clicking the pencil symbol, and check the "Visible" check-box. Click "Submit" to save the change. The "Visible" check-box makes the added problem visible at the frontend.

To hide a problem from the system frontend, the following steps needs to be executed:

1. Choose "Problem" in the topmost main menu.
2. Select the problem that is to be hidden.
3. Tick on the "Visible" check-box.
4. Click "Submit" to save changes.

Note! Deleting problems should not be done, as it may leave dangling submissions in the database.

D.4 Checker Example, Simple Diff

The checker below is used if a simple diff between expected and actual output is needed.

```
#include <stdlib.h>
#include <iostream>
#include <sstream>
using namespace std;

int main(int argc, char* argv[]) {
   if (argc != 4) {
      cerr << "wrong number of arguments!" << endl;
      return -1;
   }
   stringstream ss;
   ss << "wdiff -3 " << argv[2] << " " << argv[3] << " > /dev/null";
   int retval = system(ss.str().c_str());
   if (retval != 0)
      return 1;
   return 0;
}
```

D.5 Checker Example with Goodness

The checker below is used for the “Vertex Cover”-problem and shows how to output the goodness value “Cover size”, which is defined when adding the problem to the database.

```cpp
// C++ Vertex cover checker for exercise-3
#include <iostream>
#include <fstream>
#include <limits>
#include <assert.h>
#include <algorithm>
#include <set>
#include <vector>

using namespace std;

// Rudimentary edge structure
struct Edge {
    int u, v;
};

// Graph structure
struct Graph {
    // Number of vertices (v) and edges (e)
    int v, e;
    // A vector of edges
    std::vector<Edge> edges;
    // A vector of vertex ids
    std::vector<int> vertices;
};

// Struct for keeping track of a vertex cover and its quality
struct Solution {
    // A vertex cover and associated fitness value
    std::vector<int> cover;
    double fitness;
};

// Checks to see if the argument solution is a valid vertex cover
bool validVertexCover(Graph graph, Solution current) { ... }

// Driver program
int main(int argc, char* argv[]) {
    // Check argc
    if (argc < 3) {
        cout << "Give input filename and solution filename!" << endl;
        return -1;
    }
```

```
// Open input file
ifstream inputFile(argv[1]);
if (!inputFile) {
    cout << "No input file found" << endl;
    return -1;
}
// Open solution file
ifstream solutionFile(argv[2]);
if (!solutionFile) {
    cout << "No solution file found" << endl;
    return -1;
}
string code;
int c, u, v, N, M;
Graph graph;

while (inputFile >> code) {
    if (!code.compare("c")) { // Skip comments
        inputFile.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
    } else if (!code.compare("p")) {
        // Read number of vertices into N and number of edges into M
        inputFile >> N >> M;

        graph.v = N;
        graph.e = M;

        continue;
    } else if (!code.compare("v")) {
        inputFile >> c;

        graph.vertices.push_back(c);

        continue;
    } else if (!code.compare("a")) {
        inputFile >> u >> v;

        Edge edge = {u, v};
        graph.edges.push_back(edge);

        continue;
    } else {
        // Should never get here with the correct input
        assert(false);
    }
}
// Check student solution //
// Validate header
string header;
solutionFile >> header;
```

```
    assert(!header.compare("s"));
    // Read in list of vertices and validate that it is a vertex cover
    std::vector<int> vertices;
    int vertexId;
    while (solutionFile >> vertexId) {
        vertices.push_back(vertexId);
    }
    Solution candidate = {vertices, 0.0};
    assert(validVertexCover(graph, candidate));

    // Output for CMB
    cout << "OK" << endl;
    // Goodness value ("Cover") is second output from the checker
    cout << candidate.cover.size() << endl;
    return 0;
}
```

Appendix E

System Setup

This chapter will go into depth of the Climbing Mont Blanc system and setup information. The goal of this chapter is to complement the setup and handover instructions given by Follan and Støa [FS15]. Section E.1 proposes a uniform code folder structure in the CMB system, introduced by Follan and Støa on the development and production servers of CMB. The purpose is to allow for quicker setup of the system for local development, but also to quickly setup a new development or production server. Sections E.2, E.3, and E.4 explain the setup of the frontend, server, and backend respectively for both a new CMB instance and local development. The sections focus is on summarizing, complementing and gathering some of the setup and handover information written by Follan and Støa [FS15]. This will hopefully provide a complete and quick reference documentation of the CMB setup for new developers.

E.1 Folder Structure

The folder structure for the frontend- and server-code should be equal to the folder structure at development and production servers of CMB. It is recommended to keep the folder structure when developing locally. Having identical folder structures might reduce confusion and bugs that comes up when developing code, as some environment variables in the system depends upon the folder structure. It is much easier to set up the system as well, as little modification is needed to the configuration files to make the system work locally. The setup information is also much easier explained with a uniform folder structure. The proposed folder structure is shown in Figure E.1. When explaining the setup information, it is assumed that the folder containing the directories in Figure E.1 is called *cmb*.

The folder structure equals the folder structure at the CMB development and production server. On a CMB development or production server the folder *cmb_board* would not be present, as this folder contains the code for the CMB backend and would instead be present on the backend board. The folder structure for the

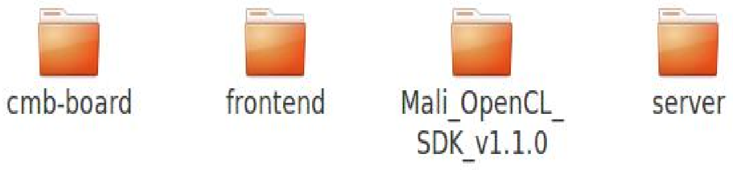
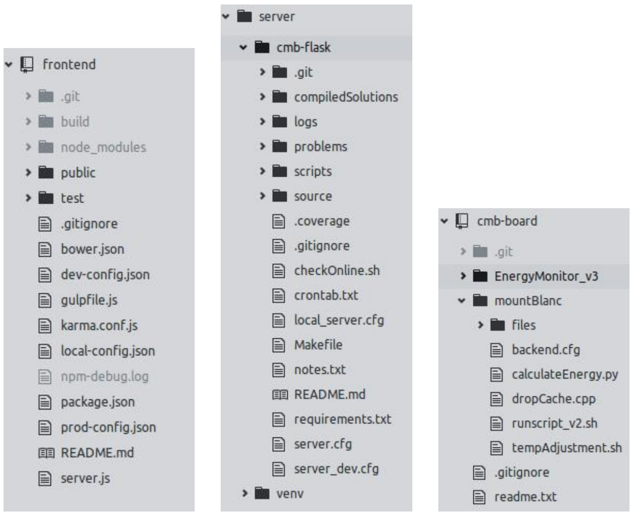


Figure E.1: Workspace folder structure



```
frontend
  .git
  build
  node_modules
  public
  test
  .gitignore
  bower.json
  dev-config.json
  gulpfile.js
  karma.conf.js
  local-config.json
  npm-debug.log
  package.json
  prod-config.json
  README.md
  server.js
```

```
server
  cmb-flask
    .git
    compiledSolutions
    logs
    problems
    scripts
    source
    .coverage
    .gitignore
    checkOnline.sh
    crontab.txt
    local_server.cfg
    Makefile
    notes.txt
    README.md
    requirements.txt
    server.cfg
    server_dev.cfg
  venv
```

```
cmb-board
  .git
  EnergyMonitor_v3
  mountBlanc
    files
    backend.cfg
    calculateEnergy.py
    dropCache.cpp
    runscript_v2.sh
    tempAdjustment.sh
  .gitignore
  readme.txt
```

(a) Frontend (b) Server (c) Backend

Figure E.2: Folder structure

frontend, server and backend code are all shown in Figure E.2. As mentioned, the code is available at Bitbucket via git, and the folder structure should be easy to setup after repository access is granted. The reader is hereby warned that a different folder structure changes the setup information. An overview of the result of the set up of local, development and production CMB system is shown in Figure E.3. Note that the backend setup is equal in each environment, and is therefore not included in the figure.

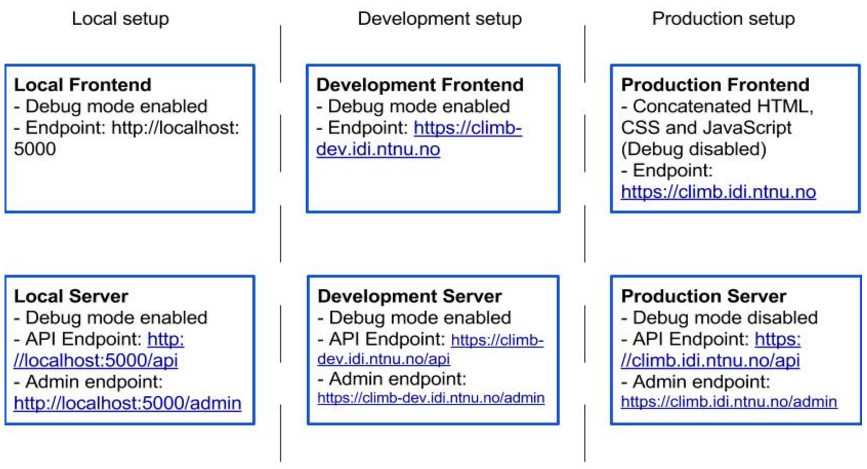

Local setup
Development setup
Production setup
Local Frontend
- Debug mode enabled
- Endpoint: http://localhost:5000
Development Frontend
- Debug mode enabled
- Endpoint: https://climb-dev.idi.ntnu.no
Production Frontend
- Concatenated HTML, CSS and JavaScript (Debug disabled)
- Endpoint: https://climb.idi.ntnu.no
Local Server
- Debug mode enabled
- API Endpoint: http://localhost:5000/api
- Admin endpoint: http://localhost:5000/admin
Development Server
- Debug mode enabled
- API Endpoint: https://climb-dev.idi.ntnu.no/api
- Admin endpoint: https://climb-dev.idi.ntnu.no/admin
Production Server
- Debug mode disabled
- API Endpoint: https://climb.idi.ntnu.no/api
- Admin endpoint: https://climb.idi.ntnu.no/admin

Figure E.3: General Overview of the Different Setup Options

E.2 Frontend Setup

Figure E.2a shows the structure of the frontend code, and will be relevant in this section. Acquire the frontend code and create the folder structure shown in figure E.2a before starting setup. This section gathers the setup and handover information from the master thesis of Follan and Støa [FS15], and is rewritten according to the proposed folder structure to make the setup easier.

CMB Frontend Setup

To build the frontend run the following command in the *frontend/* directory:

```
npm install
```

The command will install all the required packages with `npm` [NPM]. The required packages is listed in a file called *package.json*. After running the command, the required packages will be saved in a folder called *node_modules*. To build the system as it is used on the production server, the build tool `gulp` [GUL] is used with the following command in the *frontend/* directory:

```
gulp prod
```

The command will concatenate the HTML, CSS and JavaScript files which makes the code more efficient, and lowers the number of requests made when fetching data to the browser. However, since this concatenation reduces the number of source files, the code will be harder to debug. After running the commands, the frontend should be available at the URL `https://climb.idi.ntnu.no`, but keep in mind that it will not function properly before the server has been set up. The frontend will assume that a server API is available at `https://climb.idi.ntnu.no/api` to fetch data, which requires a running server as explained in section E.3. To build the development version instead, stay in the *frontend/* directory and run:

```
gulp dev
```

The command will not concatenate the HTML, CSS and JavaScript files, which will make it easier to debug using developer tools in the browser. The frontend should now be available at `https://climb-dev.idi.ntnu.no`. The development frontend also requires a running server with an API available at `https://climb-dev.idi.ntnu.no/api`, as explained in section E.3. The URLs can be changed by modifying the files *prod-config.json* or *dev-config.json* for the production and development server respectively.

Local Setup

First, as above, run the following command in the *frontend/* directory:

```
npm install
```

The command will install all dependencies from listed in *package.json*. To build the frontend for local development, run the following command in the *frontend/* directory:

```
gulp
```

The command will build the project as it is done on the development server, but the website is instead available at `http://localhost:5000`. This means that a local version of the server should also be run (described in Section E.3). The command will also watch for changes in the code, and for every change it will run the unit tests and rebuild the code. All gulp tasks are defined in the *gulpfile.js*, and new tasks can be added there. New code or modifications to existing code should be done in the *public/* folder.

Unit Tests and Linter

To make the unit tests run correctly, all `bower` [BOW] dependencies must be installed. This is done by running the following command in the *frontend/* directory:

```
bower install
```

This will install all dependencies from the *bower.json* file in the bower components folder. To run the tests, enter the *frontend/* directory and run the tests through the *gulp* task:

```
gulp test
```

When the tests are run, a directory called *coverage/* is automatically created within the *frontend/* directory. This directory contains an HTML report of the test run.

To run the `jshint` [JSH] linter manually, run the following *gulp task* in the *frontend/* directory:

```
gulp jshint
```

The command will report linter errors if any.

E.3 Server Setup

The directory structure for the server can be seen in Figure E.2b. Acquire the server code and create the folder structure in Figure E.2b before starting setup. The paragraphs ”Installation and Configuration”, ”Database Setup and Migration” and ”Unit Tests and Linter” are setup information created by Follan and Støa. The paragraphs are rewritten and complemented using the proposed folder structure to make the setup process easier.

Virtual Environment and Dependencies
A Python Virtual Environment (VE) [VIR] should be installed and used when developing and running the server code. When installed, create a new VE by running the following command in the *server/* directory:

```
virtualenv venv
```

The command will create the folder structure *venv/* within the *server/* directory. One need to activate the VE to install packages and use the packages installed within it. The Unix command *source* is used to activate the VE. An example from the *server/* directory is:

```
source ./venv/bin/activate
```

To install all required packages needed on the server, activate the VE and use `pip` [PIP] to install the required packages within the VE. All required packages is listed in the *requirements.txt* file, and is installed by running the following command from the *server/cmb-server/* directory:

```
pip install -r requirements.txt
```

The above command will install all required Python dependencies needed for correct execution of the server. Installation of requirements needs to be done both when developing locally and when setting up a new CMB server.

To deactivate the VE, execute the following command anywhere:

```
deactivate
```

Installation and Configuration

To correctly compile the uploaded solutions on the server, a *Mali OpenCL SDK* [MAL] library is needed. It is recommended to install it as shown in Figure E.1. It is also a requirement to create a directory called *workspace/* within the extracted folder of the Mali OpenCL SDK. The CMB server also needs OpenMP 4.0, gcc-4.9 and g++-4.9 to compile the uploaded programs correctly. The following steps installs the required compilers and libraries:

```
sudo apt-get-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-4.9 g++-4.9
sudo apt-get install build-essential
sudo apt-get install libmysqlclient-dev
```

Uncomplicated Firewall (UFW) [UFW] and Fail2Ban [?] are installed and configured by running the commands below:

```
sudo apt-get install fail2ban
sudo ufw allow 80
sudo ufw allow 443
sudo ufw enable
```

UFW should already be present, and does not need installation. The second and third command above allows requests to port 80 and 443. That is, all HTTP and HTTPS request should be allowed to pass to the server. Fail2Ban should work out of the box.

Nginx [NGI] can be installed on development and production servers in the following manner:

```
sudo apt-get update
sudo apt-get install nginx
sudo service nginx stop
```

Make sure to remove the default Nginx configuration if present in the directory */etc/nginx/sites_enabled*. The server code repository contains the Nginx configuration files *prod_nginx.conf* and *dev_nginx.conf* for the production and development server setup respectively. Copy the Nginx configuration file into the directory */etc/nginx/sites_available*. Also, create a symbolic link to the file in the directory */etc/nginx/sites_enabled* and restart Nginx:

```
sudo ln -s /etc/nginx/sites_available /etc/nginx/sites_enabled
sudo service nginx stop
```

Make sure to also acquire a valid Uninett certificate if setting up a production server.

Some environment variables need to be set up to configure a CMB server correctly. Firstly, the Unix environment variable *APPLICATIONS_SETTINGS* need to be set. This environment variable points to a file that contains application specific variables, depending on if the server is either a production, development or a local server. The files *server.cfg*, *server_dev.cfg* and *local_server.cfg* present in the directory *server/cmb-flask/* represents the application settings for a production, development or a local server respectively. For example, the IP address of a backend board, the path for the Mali OpenCL SDK, the database URI and the server port is all present in the configuration files among other variables. Section E.5 gives an example of how the file *local_server.cfg* might look like. It is very important to enter these values correctly.

The next environment variables that need to be set are *CMB_MAIL_USERNAME*, *CMB_MAIL_PASSWORD*, *CMB_SECRET_KEY* and *CMB_TOKEN_SECRET*. The variables *CMB_MAIL_USERNAME* and *CMB_MAIL_PASSWORD* are used to send an email to the CMB administrators when an error occurs, which is crucial if setting up a production environment. The variables *CMB_SECRET_KEY* and *CMB_TOKEN_SECRET* are used for session token generation, authenticating messages and more. When developing locally, these variables can stay unchanged and can be copied from the file *local_server.sh* present in the directory *cmb-flask/scripts/*. If setting up a development or production server, contact the CMB team, and they will provide the information to set these variables correctly. It is recommended to enter the variables into *$HOME/.bash_profile* or some similar file that loads the environment variables when starting up a new shell.

The environment variable *SLQALCHEMY_DATABASE_URI*[1] and *SERVER_TIMEOUT* also needs to be set in the *$HOME/.bash_profile* or some similar file which loads the environment variables. The server timeout describes the timeout of SSH commands and is currently defined to be 95 seconds.

Run the following command from the directory *server/cmb-flask/scripts/* to start either a development or production server:

```
./init_cmb start
```

The command will start the whole CMB system, including the frontend and *push.py*. Keep in mind that the system will not function correctly before a backend has been set up, as described in section E.4. The above command also allows the arguments *stop* and *restart* as well, to stop or restart the CMB system respectively. The above command is the recommended way of starting either a production or development server. To start a local server, run the following command in the *server/cmb-flask/source/* directory:

```
python server.py start
```

[1]The database URI describes which database to connect to. Example with MySQL: mysql://<username>:<password>@<host>/<db_name>.

This will start the server without *Gunicorn* [GUN]. Running the server without Gunicorn makes it easier to debug, as only a single instance of the server is running. However, it is possible to launch the CMB locally with the *init_cmb* script locally as well if wanted. No matter the method of running the server, it will create a callable API at endpoint `https://localhost:5000`. It is important to initialize the database (see below) before running the server the first time.

The following lines can also be added in the file */etc/rc.local* to automatically start the system when booting the server:

```
su climber -s /bin/bash -l -c
    "/srv/climber/cmb/server/cmb-flask/scripts/init\_cmb.sh start" >
    /dev/null
```

Database Setup and Migration

To clear and initialize a database for a new server, run the following command in the directory *server/cmb-flask/*:

```
python init_db.py
```

If there is a modification to the database models (read table schema), one needs to *migrate* [2] the database. Migration happens by creating a migration script, generated by running the following command in from the *server/cmb-flask/source/* directory:

```
python server.py db migrate -m "some message."
```

This will create a new directory within *server/cmb-flask/source/* called *migrations* if not present, with the auto generated migrations script within. To launch the migration script and alter the database table schema, stay in the directory *server/cmb-flask/source/* and run:

```
python server.py db upgrade
```

On the production and development servers, this is done automatically through Jenkins when there is a change to the database tables. The migration script is also added to the git repository so that the above step can be executed manually.

The production and development databases are provided by the IDI department [IDIa]. If developing locally, remember that MySQL needs to be installed and a local database needs to be set up. Remember to set the correct database URI in the *$HOME/.bash_profile* or similar file as described above.

[2]Migration is the task of updating or reverting a database schema while trying to preserve the data that might be present in the database.

Unit Tests and Linter

After installing and activating the virtual environment above, make sure all required packages from *requirements.txt* are installed. From the directory *server/cmb-flask/source/*, run the following command:

```
nosetests --with-coverage
    --cover-package=admin,routes,cmb_utils,server,database tests/*.py
```

This will run automatic unit tests with `nose` [NOS] for all tests present in the directory *server/cmb-flask/source*. It will also generate a cover report, with an extensive overview of the code covered by the tests. If developing a new test, it might be useful running just a single test. To run a single test, execute the following command the *server/cmb-flask/source/* folder:

```
nosetest tests/problems_test.py
```

This will run the tests present in the *problems_test.py* file. One can simply run another test set by changing the file name to another file present in the *server/cmb-flask/source/tests/* directory.
To run the `flake8` linter, stay in the *server/cmb-flask/source/* directory and run:

```
flake8 .
```

The terminal window will report potential errors.

E.4 Backend Setup

This section explains the setup of a new Odroid-XU3 board with the folder structure equal to the one in Figure E.2c. The folder structure is relevant when setting up the CMB board code as explained below. The paragraph "CMB Setup" contains instructions and information made by Follan and Støa, which is further complemented (uninstalling `lightdm` [LIG]) and rewritten to make setup easier.

Odroid-XU3 Setup

A new Odroid-XU3 board should have a fresh installation of Xubuntu installed available at the Odroid website [ODR]. The Xubuntu installation information given here is inspired by the information stated at the Odroid website [ODR]. The Odroid-XU3 can either boot from a MicroSD card or a special module called eMMC module. The eMMC module needs to be connected to a eMMC module reader as seen in the Figure E.4 to be able to flash. A Unix operating system is assumed used when flashing the OS image onto an eMMC module or a MicroSD. For flashing of OS images onto MicroSD like media using Windows, refer to the Odroid website [ODR]. After downloading a new Operating System image, flash the connected MicroSD card or eMMC module with the following commands within the folder where the image is downloaded:

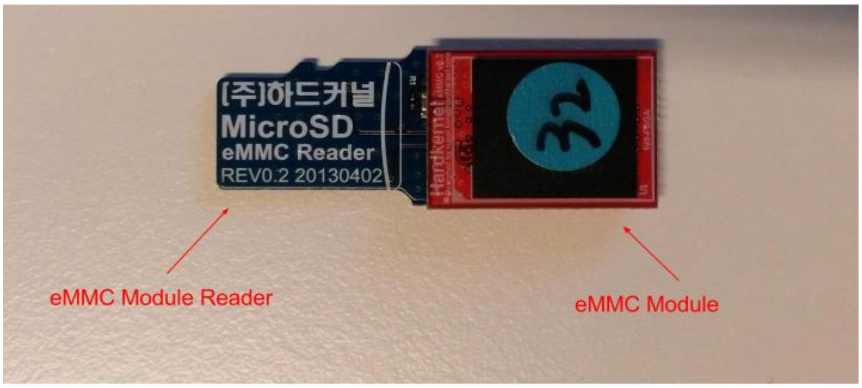


Figure E.4: eMMC Module and Reader

```
unxz some-xubuntu-image-file.img.xz
sudo dd if=/dev/zero of=</dev/path/of/card> bs=4M conv=fsync
sudo dd if=<some-xubuntu-image-file.img> of=</dev/path/of/card> bs=4M
    conv=fsync
sync
```

The path of the MicroSD card or eMMC module can be found by monitoring the directory */dev/* before and after connecting the device. The connected device will appear as sdX, where X is some alphabetical character. As mentioned, CMB uses the EnergyMonitor program to measure energy consumption, and it is important to check that the downloaded OS image supports the EnergyMonitor program.

After installing the OS, connect either the MicroSD or eMMC module to the board[3]. Then boot and connect the Odroid-XU3 to a monitor either through mini-HDMI or a DisplayPort. An automatic login will happen to a user called *odroid*. Open a terminal window and create two more users by running the following commands:

```
useradd climber
passwd climber
useradd worker
```

This creates two users, *climber* and *worker*. When prompted for a password for the *climber* user, enter the password given to you by the CMB team. These two users are needed to execute commands and run programs through the server. Additionally, enter the following line to the end of the file *sudoers* located in the directory */etc/*:

[3]Check out `http://odroid.com/dokuwiki/doku.php?id=en:xu3_bootmode_configuration` on how to toggle between booting from eMMC module and MicroSD.

```
...
climber   ALL=(worker) NOPASSWD: ALL
...
```

The line will make sure that the *worker* user have the privileges to execute the uploaded programs. The command *visodu* should be used to add the line to the *sudoers* file.

The backend should install OpenSSH and OpenSSH Server if not already installed. The two services can be installed and configured to CMB by executing the following commands:

```
sudo apt-get install ssh-client
sudo apt-get install ssh-server
ssh-keygen -t rsa
ssh-copy-id username@ip-to-server
```

When prompted for something, just hit enter. After executing the commands above, login to the board can happen without entering a password. This will make sure that the server does not get prompted for a password when it needs to execute scripts at the backend. You should not be prompted for a password when logging into the board from the server after executing these commands.

Uncomplicated Firewall (UFW) and Fail2Ban are installed much the same way as on the server. Execute the following commands to install and configure both services:

```
sudo apt-get install fail2ban
sudo ufw allow from 127.241.0.0/16
sudo ufw enable
```

The board should not be accessible from outside the NTNU network, which this UFW configuration ensures.

CMB Setup

Follan and Støa reported the CMB backend setup in their Master thesis [FS15]. The commands to setup the CMB backend code are repeated here, and extended with the removal of `lightdm` [LIG]. Log in as *climber* at the board through SSH and fetch the backend code from Bitbucket (clone the repository with git), and run the following commands to correctly setup the CMB backend:

```
# Link binary
sudo ln -sf /lib/ld-linux-armhf.so.3 /lib/ld-linux.so.3
# programs and packages used by runscript_v2.sh, EnergyMonitor and
calculateEnergy.py.
sudo apt-get install python-scipy time qt4-default libqwt-dev
# install g++ and gcc 4.9 to support OpenMP 4.0
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-4.9 g++-4.9
sudo apt-get install build-essential
# add following line to /etc/rc.local. This script is run automatically on
    boot,
# and is needed for tempAdjustment to read current temperature.
# Permissions are reset when rebooting.
chmod +r /sys/devices/10060000.tmu/temp
# These are needed by the EnergyMonitor_v3.
# Make sure executable name is "EnergyMonitor"
cd cmb-board/EnergyMonitor_v3
qmake
make
# Remove lightdm for accurate energy readings
sudo apt-get purge lightdm
# compile dropCache.cpp to dropCache and make it an auxiliary executable.
# dropCache is responsible for clearing the cache before running a program.
cd ~/cmb-board/mountBlanc
g++ -O2 dropCache.cpp -o dropCache
chmod 4710 dropCache
```

Download the Mali OpenCL SDK v1.1 and copy the folders *common*, *lib* and *include* into the directory *cmb-board*. The folders can also be transferred from the server.

E.5 Local Server Configuration File

The configuration file presented here assumes that the project have been structured as proposed in section E.1. An example configuration file is listed below, which only requires one to find the path to the *cmb/* directory and the executing backend IP address.

```
SERVER_PORT=5000
BOARD_IP="THE_BOARD_IP_ADDRESS"
MALI_DIR="<path-to-cmb>/cmb/Mali_OpenCL_SDK_v1.1.0"
FLASK_DIR="<path-to-cmb>/cmb/server/cmb-flask/"
FRONTEND_DIR="<path-to-cmb>/cmb/frontend/"
UPLOAD_FOLDER="<path-to-cmb>/cmb/server/cmb-flask/problems"
MAIL_SERVER='smtp.gmail.com'
MAIL_PORT=465
MAIL_USE_TLS=False
MAIL_USE_SSL=True
GUNICORN_LOG_LEVEL="debug"
VERSION="dev"
```