

## 0.1 Foreword

Originally inspired by the Nordic Collegiate Programming Contest (NCPC), it has been held at NTNU every spring since 2007. The format is a five-hour contest with competing teams consisting of one, two or three contestants. A team of volunteer judges write the problems and answer clarification requests during the contest, while another team hands out balloons for each solved problem. Usually a rather hectic affair, it is extremely important that everything is well prepared. The number of teams is often more than 100, with the record being 162 teams in 2011

The contest system that verifies solutions is at the heart of the contest when it is in progress, and needs to be working perfectly at all times. The system must handle several submissions per second, while verifying that each one is correct and runs within the set resource limits. Submissions must show up on the high score list, and when problems are solved the team handing out balloons must be notified. In addition to this there were a lot of other functional requirements having to do with the bureaucracy of organizing the contest

A requirement was that new features could be easily added in the future, and the code was written with this in mind. The project will now become open source, and all programming contest enthusiasts will soon be able to request and implement their desired features

All aspects of this project have been pleasing and delightful for us. The team has exceeded all our expectations and their system will be used for years to come.

## 0.2 Preface

Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing. Designing a system for eager students to hone their skill in the heart of computing has been a true joy

Our group never wanted to settle for adequacy and mere requisiteness. For the past few months, weve taught ourselves a new programming language and framework and used advanced development frameworks - while tackling many social and technical conflicts.

We have ve proven how Ambition is a dream with a V8 engine, as Elvis Presley once said.

The group would like to thank our eager customers, Finn Inderhaug Holme, Christian Chavez and Christian Neverdal Jonassen for their time to meet us and provide constructive feedback. We also owe a big thanks to our supervisor, Hong Guo, for constructive criticism and reflections; without which, we would not ascertain the peak of our own potential

# Chapter 1

## Introduction

### 1.1 About the course

Our group and assignment has been delegated as part of the course IT2901: “Informatics Project II” at NTNU. The work covers 15 course credits, equivalent to a 50% work position for one academic semester. IT2901 is offered only to those that are enrolled on the NTNU’s informatics BSc programme.

The primary purpose of the course is to let students apply their knowledge from other courses. This is rendered through a project for a real customer. The students have to communicate independently with their client, and deliver a software product that answers the client’s needs.

Grades are based on the satisfaction of the customers and an evaluation of the development process. The latter will be reviewed through written reports and timesheets, as provided in this document. Furthermore, it is important that students have met the given deadlines and documented their work in a structured manner.

### 1.2 The Group

The team consists of six members. All the members of the group are completing their BSc degree in Computer Science from NTNU in 2014. We had prior experience working together, and knew each other well. With many shared courses and similar interests, the team are all at a somewhat similar level of competence. However, we have different areas of expertise, and exploiting this has been a key to success on previous occasions. For a detailed description of each member, see the listing below.

**Anders Sildnes** Throughout his BSc, Anders has been taking courses related to algorithms and program security. Apart from his studies, he is developing for Engineers without Borders NTNU and spending time with open-source projects and other Linux tools.

**Eirik Fosse** Eirik has a primary interest in artificial intelligence and machine learning. In the course of his bachelor's degree he's focused on programming, mathematics, and evolutionary simulation.

**Filip Fjuk Egge** While achieving his degree, Filip has taken courses focused on a path related to system development and security. He has a varied education and knowledge on different aspects of computer science.

**Haakon Konrad William Aasebø** Haakon has selected disciplines related to mathematics and algorithms. Apart from being a student at NTNU he is playing football at NTNUI in the third division.

**Håkon Gimnes Kaurel** During his time at NTNU, Håkon has been keeping a primary focus on courses related to programming and the intersection between hardware and software. He's also got experience as an app developer, and has extensive knowledge of the GNU/Linux operating system.

**Tino Lazreg** Tino has been taking courses related to different aspects of software engineering, like programming, system architecture, human-machine interaction. Besides doing a BSc, Tino also works as a student assistant in a human-machine interaction course on NTNU.

## 1.3 About the Customer

Our customer is IDI Open. They are responsible for the annual programming contest mentioned in 1.2. Christian Chavez is our main contact for the project, but his two colleagues, Christian Neverdal Jonassen and Finn Inderhaug Holme, were also available for questions. They are all students of computer science at NTNU.

## 1.4 About the Contest

IDI Open is a programming contest where teams of up to three people meet and solve programming problems of various difficulty. The contest lasts five hours, and the objective is to solve as many problems as possible. The contest is open for all types of programmers, from students of all grades to professors and other professionals from the IT industry. Various prizes are given to the teams based on their performance. There are usually 8-12 problems in a contest. To make the competition fun for everyone, there are typically some problems that are easy enough even for novice programmers to handle. The main objective is to solve the highest amount of problems in the shortest amount of time.

## 1.5 Stakeholders

Our stakeholder fall into two different categories: the ones involved in the competition, and those involved in the course.

### 1.5.1 Course

**Supervisor** The supervisor's job consists of guiding and helping us through this project. This aid was primarily focused on the development process and the writing of this report. The supervisor tries to ensure that the developers communicate properly and have a structured approach to developing the end product. To verify this, we have had bi weekly status reports delivered to the supervisor, as well as regular meetings.

**Examiner** The examiner(s) is responsible for determining our final grade. Unlike the other stakeholders, we have not communicated with the examiner throughout the development process. Though, the examiner has got access to all the documents the supervisor has got access to.

### 1.5.2 Product

**IDI Open** The project's primary stakeholders. They are the host of the competition in which our product was used. Their inclusion in this product comprised all aspects of our project.

**Judges** The judges are hired by IDI Open to supervise the competition, service contestants and create problem sets. They will rely on our end product achieve the mentioned tasks. Throughout the process they have given feedback to our customers, IDI Open, about our product. Naturally, the judges are important to the contest, so it is important that they are satisfied with the software they have to use.

**Developers** The developers are responsible for satisfying all other parties. Similar to the customer, our involvement in this project is total.

**Maintainers** As IDI Open is an annually recurring event, our end product, if successful, will be used for many years in the future. At a point, we assume the code will need to be replaced or modified. Assumably, there will be another developer team to do this. As such, the quality of of our product will impact them.

**Sponsors** Different companies sponsor each contest. In exchange for money and services, the sponsors get exposure through ads on the website and get to give a short presentation during the awards ceremony. Naturally, the sponsors want to associate their name with a successful product. Therefore, the sponsors rely on that contests are successful - this is heavily based on our product.

**Contestants** The actions of contestants are all through our software; our product will be their medium to take part in IDI Open. Reliability and usability is key to keep the contestants happy. The contestants also gave feedback to the customers about their user experience. Thus, how satisfied the contestants are impacts the developer's evaluation.

## 1.6 Goals

The goal of the project is to upgrade and improve the existing system used in IDI Open. We were given sole responsibility for our project; no other team or organization of developers has had responsibility for our solution. This gave us inspiration to do the best we could, and to give the customer something both we and they could be proud of for many years to come. And if the product is good enough it would hopefully also be used in larger programming competitions, maybe even international ones.

## Chapter 2

# Task Description

### 2.1 Task Description and Overview

The first step in our development process was to get a brief overview of our complete system. To do this, we have followed a conventional style of designing UML use cases together with a textual description. From reading this chapter, the reader should be able to understand how our end product works. Also, reading this chapter will be important to understand the other chapters in this report. The assumptions and constraints that affected our process are also discussed in section X.X.

#### 2.1.1 Assignment

According to our customers, “IDI Open”’s previous solution was cumbersome to use. Our assignment was to create a replacement system that would be easier to administer. This included replacing both front and back-end systems. In replacing the old solution, we did not need to implement features that were not available from the old system.

The features of the old solution, in a nutshell, are given below. A more detailed overview will be given in section X.X.

- Website to inform about the contest to the public
- Team-registration and scoring
- Ability for users to upload code, that would be compiled and executed by the server

We were given access to the code for the old solution. The customers felt that this code was cluttered, but we could re-use components wherever we wanted. However, it was important that we did this in an uncluttered way, such that other developers could easily understand the new solution.

### 2.1.2 Branding our Product

Since we were delivering an end product to a real customer, we wanted to present ourselves as a real company. We chose the name “GentleCoding” as our representative name. This was used to name our repositories, email lists and other medium communicated with external parties.

The term “Gentle” is supposed to represent our calm approach to problems. It is also similar to “Gentleman”, which reminds of quality and good conduct . Furthermore, it is easy to interpret and remember.

Since we were developing a new system, we also wanted to brand our product. We wanted to keep it logical and simple, so we decided on “GentleIDI”. Consequently, GentleIDI will be used to refer to our end product throughout the rest of the report.

### 2.1.3 Assumptions and Constraints

To define what is satisfactory, we have made some assumptions and defined some constraints. Table X.X should make it easier to understand how we have reasoned our system design.

Do note that the implications in table X.X were not necessarily upheld. Rather, they were used as initial bounds to permit leeway. For example, imposing that third party plugins will speed up development does not mean that we would always prioritize software re-use.

**Roles and Their Definitions** Usergroups Within the application-domain of “gentleidi” there are different groups of users. Each group has different levels of access control, and once a user is made a member of that group, they inherit those rights. A user may have membership in all groups. A privileged user is someone who is manually given elevated permissions. Table X.X shows the different roles and their available actions. Further elaborations on each group will also be given in later sections, but table X.X should suffice for an overview.

Table X.X: Usergroup overview

Service-providing Units

Another way of viewing the task description in section X.X, is to say that our solution needs to do three actions: serve web-content, store data and execute user-submitted code. Since each of these operates with different protocols, we will think to our solution as composed of three different systems. These are described in figure X.X.

The only entity that end-users interact with is the webserver. This is done through HTTP messages, which, if necessary, are relayed to the execution nodes and/or database. **UML Use Cases** We need one page each for privileged, registered, and non-registered users. That is, one interface for administrative users, one for contestants, and one for non-registered viewers. From each of these three, we defined use case scenarios. Figure X.X and X.X models the available workflows and actions for each category of users. Table X.X describes the semantics of objects used in the diagram, which should be equivalent to the UML 2.0 standard.





Assumption/Constraint	Why	Implication
The system will be maintained by people who have experience with computers	People that are involved with any programming contest are typically programmers themselves.	User design, words and definitions can be made more technical. Error messages can be explained using computer lingo.
The system will be used and maintained for > 5 years	Customer-constraint: they do not want to spend too much time developing new products, so maintenance is preferred.	The code should be written in a modular, extensible way with clear documentation.
The customers, IDI Open, are students near/at Gløshaugen		High availability for customer meetings and reviews.
The developers will maintain a 20 hour a week work ethic throughout the project-duration of 20 weeks[TODO: update]	To finish the product (on-time)	The set of requirements should not require more than 20 hours of work per week per developer, in order to complete.
Our system should be user-friendly	Our solution features a web interface available to everyone. Ideally, any person should be comfortable with the user interface.	
Our end product will be open sourced	To ensure quality, and let other volunteers contribute to the code repository	No proprietary third party modules can be used. We cannot copyright our own material.
The final solution must run on Linux-computer	This is the choice of OS by NTNU, which is responsible for technical support and server access	Linux-compliant solution
We are allowed to use whatever third party plugin we want, as long as it is free and has no copyright-conflict	Speed up development	Speed up development

Role	Description
Admin	Privileged. An admin can modify all the available settings of the system
Judge	Privileged. Similar to an admin account, but with a limited set of actions: answering questions (clarification system), upload tasks to be solved, solutions to those tasks, upload incorrect answers (eg. answers that will provide penalty).
Functionary	Privileged. Functionaries hand out balloons when a team has solved a task. To determine what team will be given a balloon, the functionaries have their own interface with a team overview.
Team	A group of one to three contestants. A contestant is only part of one team per contest.
Contestants	A contestant has an account on the system and has the possibility to enter and compete in a contest.

Table 2.1: CAPTION

Entity	Features	Protocol
Webserver	Processes requests from contestants and teams. Also acts as a web gate interface to the execution node, both receiving and transmitting data to other execution nodes on the behalf of users.	HTTP
Execution node	A service, often on a dedicated platform, that offers the ability to compile and execute code. The execution node returns output data to the webserver.	RPC
Database	The storage unit for all user-data and logs. Also the execution node writes data to the database, and	SQL

Table 2.2: CAPTION

	Use case actor. Represents a user group
	UML generalization arrow. Used to indicate inheritance. The arrow's source, i.e. tail, represents the entity that inherits from where the arrow points to.
<<include>>	UML stereotype to represent a mandatory extension to a work-flow.
<<extend>>	UML stereotype to indicate that if certain conditions are met in a flow, the entity to which this arrow points to can extend the workflow.

The purpose of the use case diagrams is to give a clear overview of what users shall be able to accomplish from our system. Furthermore, use case diagrams are easier to communicate to external parties, such that it is easier to agree on the system's properties. The use case diagrams were used early in development to agree on the requirements specification and to communicate to the supervisor what we were trying to accomplish.

As seen in figure X.X, admins has privileges to perform the actions of any other group, in addition to their own set of actions. Thus, membership in the admin group gives a user complete control in the application domain. Furtherly, it can be noted that all usergroups have the opportunity to act as a contestant to review the website. Privileged users will are still restricted from appearing in the official high score tables to prevent them from assuming a competing role. This was to avoid the chance of any person with access to the solutions to compete.

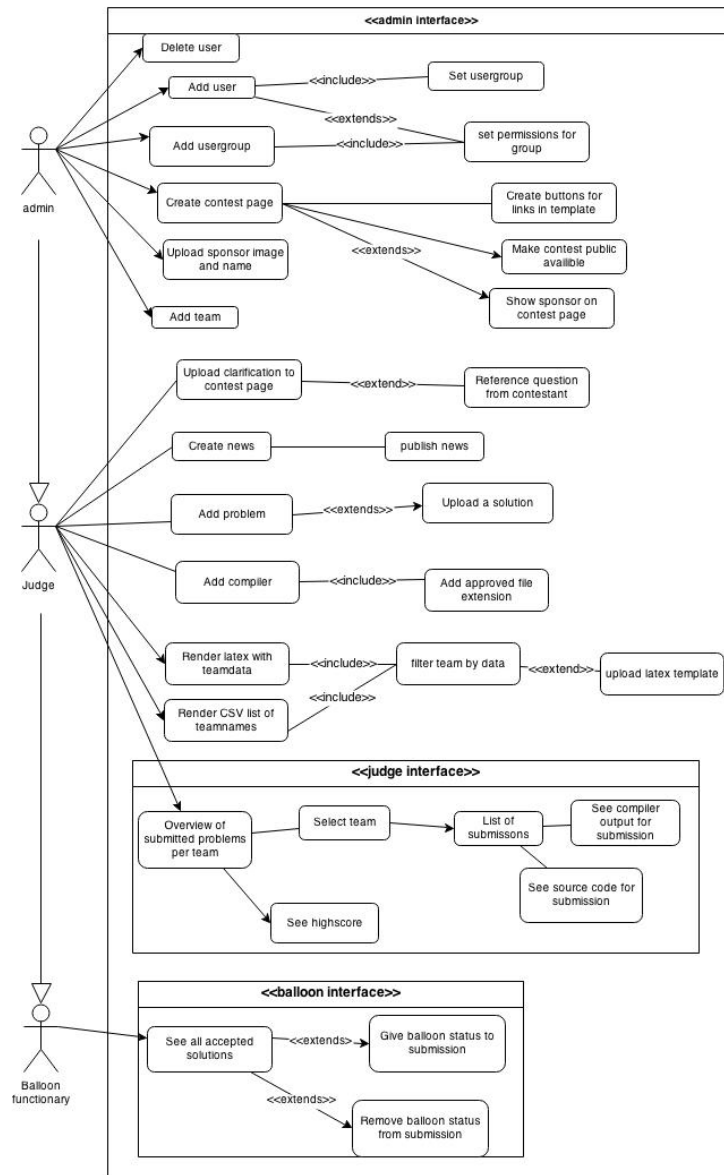


Figure 2.1: FIGURECAPTION

# Chapter 3

## Architecture

This chapter contains an overview over the architecture for the system. The first part will describe different views of the system and the second part will show the quality attributes and patterns used when developing the system.

The main parts of the system is shown in Figure x.x. Clients sends requests to the web server and receives the processed results. The execution nodes process user submissions, and updates the results to the database. Figure X.X

### 3.1 Views

We have chosen to depict the architecture using Philippe Kruchten’s 4+1 view model. [1] This is a method of describing the architecture for software-intensive systems from the viewpoint of different stakeholder by using multiple, concurrent views. We chose this model because it gives a good overview and is widely accepted by the software industry. Below are the 4 main views in the model; Logic, Process, Development, and Physical. The “+1” view is Use Cases which is addressed in [Chapter 2 Task Description and Overview]

#### 3.1.1 Logic View

Purpose:

The logical view describes the functionality of the system by breaking down requirements into classes and representing them, and their relations, through class and sequence diagrams.

Figure 6.1 shows the main classes involved in GentleIDI. Each team participates in a single contest, and consists of a predefined number of contestants. Each team also has a team leader that handles most of the administrative tasks. The team can also try to solve problems by uploading submissions.

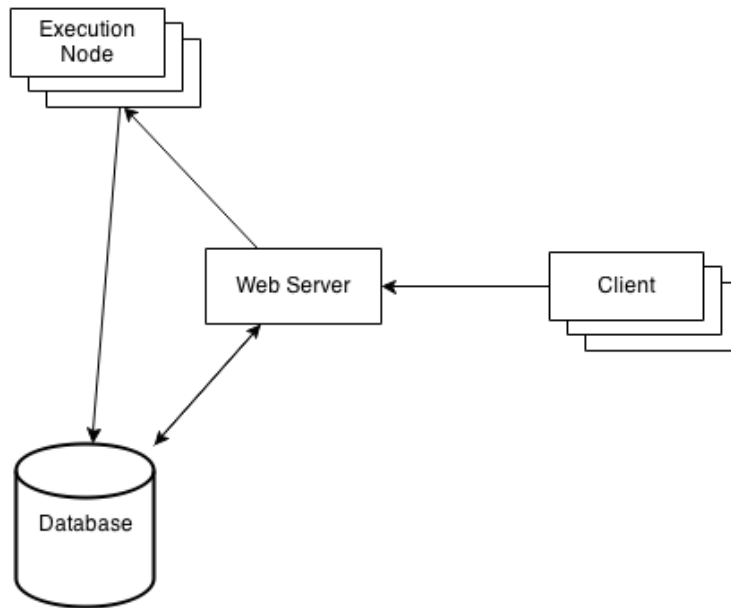


Figure 3.1: System overview

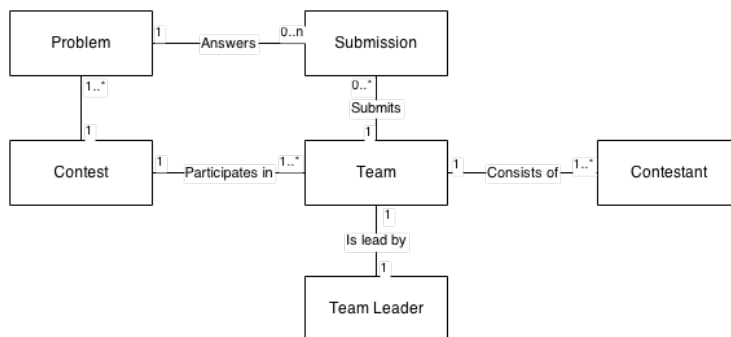


Figure 3.2: Top level class diagram

### 3.1.2 Process View

Purpose:

The process view explains the communication between different processes in the system, as well as how the system behaves in runtime.

As this system is a web application the first thing to note is that there will be concurrent users in runtime. Each user generates HTTP requests to the server, which in turn may execute database lookups for information like score tables or problem sets. When a user submits a solution the system will place it

in a queue, which decides which node the solution will execute on according to availability and load.

We will now show examples for two important parts of the application. First is the action of successfully registering a user and creating a team. See figure 6.2.

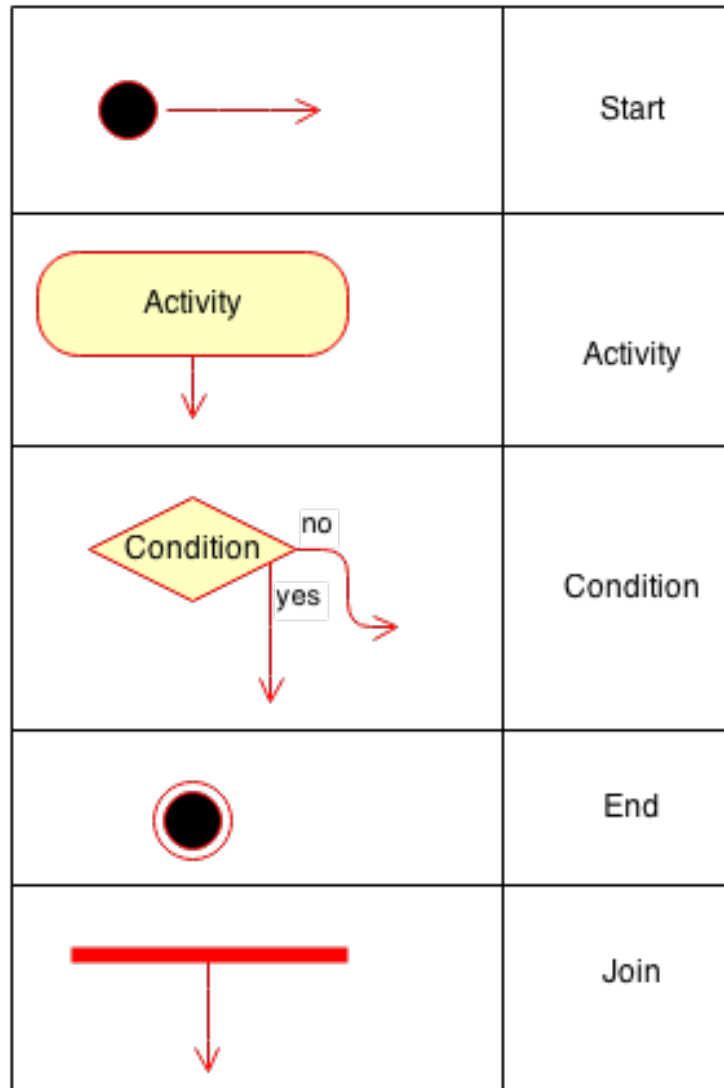


Figure 3.3: Symbology

Second is submitting a solution to a programming problem. See figure 6.3.

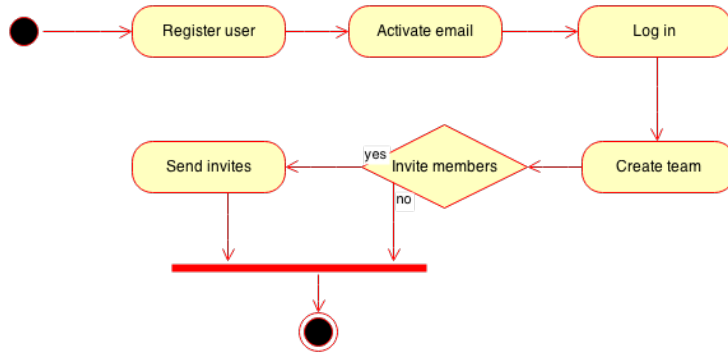


Figure 3.4: Activity Diagram for registering a user and a team

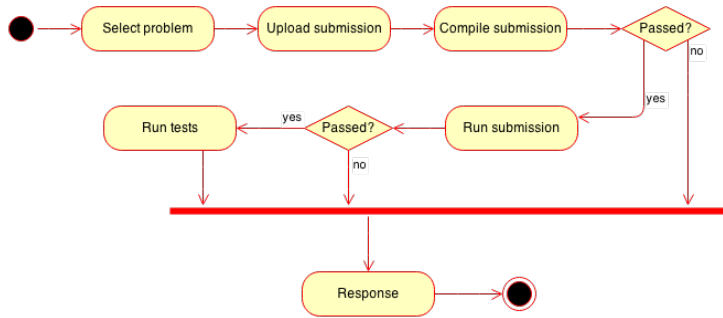


Figure 3.5: Activity Diagram for submission a solution

### 3.1.3 Development View

Purpose:

The developer view is intended for the developers. It should ease development, and focus on software module organization by packaging the software in small chunks.

We wanted a modular and maintainable system where it is easy to maintain and change specific parts of the system without changing everything. The structure of the system can therefore be divided into the following main packages: Contest, Registration, Submission, Execution, Balloon, Clarification, Admin, and Article. These packages are described in detail in chapter 8 Implementation.

### 3.1.4 Physical View

Purpose:

The physical view shows the interaction between the physical components of the system.



Physically the system is structured as a multitiered architecture. It consists of three tiers, presentation tier, application tier, and data tier, see figure 3.6. The tiers represents a physical structuring mechanism for the system infrastructure. The user is physically separate from the application and database.

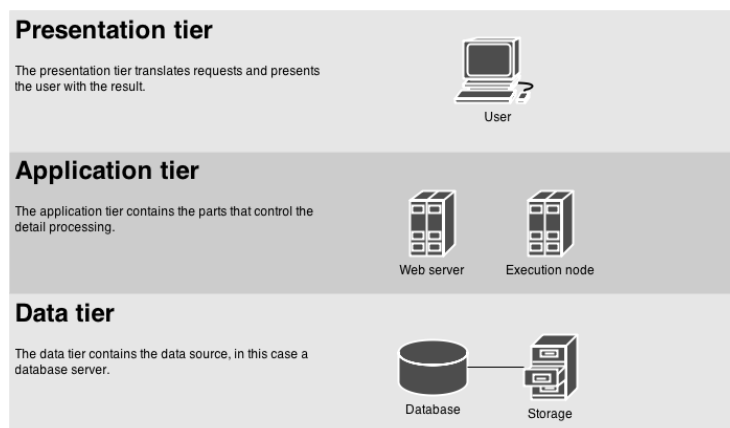


Figure 3.6: Multitier architecture

### Presentation tier

This tier presents information to the user through the public website and admin interface. It translates the web server response into web pages generated using HTML5, CSS, Ajax, and JavaScript. It sends requests to the underlying web server and renders the response.

### Application tier

The application tier contains the logical layer, it controls an application's functionality by performing detailed processing. Primarily this is done through python code, although when running solutions the file is run on an execution node through the use of built in unix commands.

This splits the application tier in two parts, the web server that serves static and dynamic content, and the execution nodes that process uploaded submissions. This division can be seen in "Application Tier" in Figure 6.4.

For the web server we use Nginx for serving static files, and as a reverse proxy for Gunicorn, the server providing dynamic HTTP content to the user. Gunicorn is the server that processes requests and returns HTTP pages. The execution nodes process submissions through a FIFO queue implemented with Celery and RabbitMQ. This provides load balancing across CPU cores and multiple nodes

in the cluster. The execution nodes also share parts of the filesystem, this is implemented with SSHFS (SSH Filesystem), and is a secure way of sharing the uploaded files across the execution nodes.

### **Data tier**

This tier includes the data control functionality. The system utilises a shared SQL database for the execution nodes and the web server. See Figure X.X This database links to the file storage on the main web server. However, the execution nodes requires some files to be shared across multiple nodes. Like explained earlier in section X.X, this is implemented with SSHFS. For more specific details see [Chapter 8 Implementation] and [Appendix ER figures]

## **3.2 Quality attributes**

### **3.2.1 Availability**

Since this software is to be used in a programming contest, it is crucial that the system has high uptime and availability. And since the contest only lasts for about 4 hours, our margin for failure is minimal. We have made an effort to account for all possible outcomes, and to safeguard the application for any errors that might occur.

### **3.2.2 Modifiability**

This is a system that we hope will be used for many years to come. With the ever changing nature of the web, the ability to adapt and improve is imperative. To accommodate this, we chose to implement our solution in Python, a language taught to most of new students of computer courses at NTNU. These are the same students that hopefully will use and continue to work on this software. To our best ability we have also tried to write and document the code in a way such that it is easy to understand and improve.

### **3.2.3 Performance**

Performance is an important aspect of every application, especially web applications. Users expect that sites loads fast. Failing to accomplish this is a sign of a bad application, at least from the user's perspective. For this reason we have focused on making our pages load as fast as possible. And since this application will be used by over 100 users simultaneously, it is also important that the servers will handle the load.

### **3.2.4 Security**

Since our application contains user data and data that should be hidden from unauthenticated users, security is another important aspect. Django provides

many security features by default, and others that can be implemented with very little effort. We also chose to enable SSL on the web server to increase security on web requests.

### **3.2.5 Testability**

When we first started out, we wanted to utilize testing during development. Testing is a way to find problems early, and before they begin to encompass larger parts of the application. But testing is also one of the most time consuming parts of the development process. In the end we did not have as much test coverage as we would like, but we feel that we covered the most important parts.

### **3.2.6 Usability**

As with any web application, we want the users of the system to accomplish their desired task, and learn the functions of the system with ease. The user should receive feedback if something went wrong or if the outcome is not clear. We also want the web pages to provide information how to use the system.

## **3.3 Patterns**

### **3.3.1 Client-Server**

Since we are making a web application we will use the Client-Server pattern. The clients connect to the server through a web interface, either the website or the admin interface.

### **3.3.2 MVC(model-view-controller)**

The front end is implemented using the Django framework and follows a rather strict implementation of MVC. Every HTTP request sent to the site is handled by a controller function, which in turn fetches the appropriate models from a database, creates a view based on the models and returns the view as an HTTP response.

### **3.3.3 Shared-Data**

The system utilises multiple execution nodes as well as a web server, through which users access data. We wanted to have a central shared database server that scales with the number of execution nodes and the amount of data.

### **3.3.4 Multi-tier**

See: 6.1.4 Physical View.

References:

- [1] Architectural Views -

### 3.4 Design

This document contains the choices made regarding the process of designing the front-end of the application, for a more technical approach see *System Architecture chapter 6*.

#### Design process

The user interface provided by the previous IDI Open system consisted of a simple web interface for reading news items, registering teams for contests, and delivering submissions. GentleIDI is intended to provide more functionality through its web interface, including but not limited to change email(requirement FC-02), supervisor(requirement FJ-11) and user management (requirements FC-01, FC-03 and FC-04). As a consequence we had two options available: reusing and extending the existing interface design, or creating our own design from scratch.

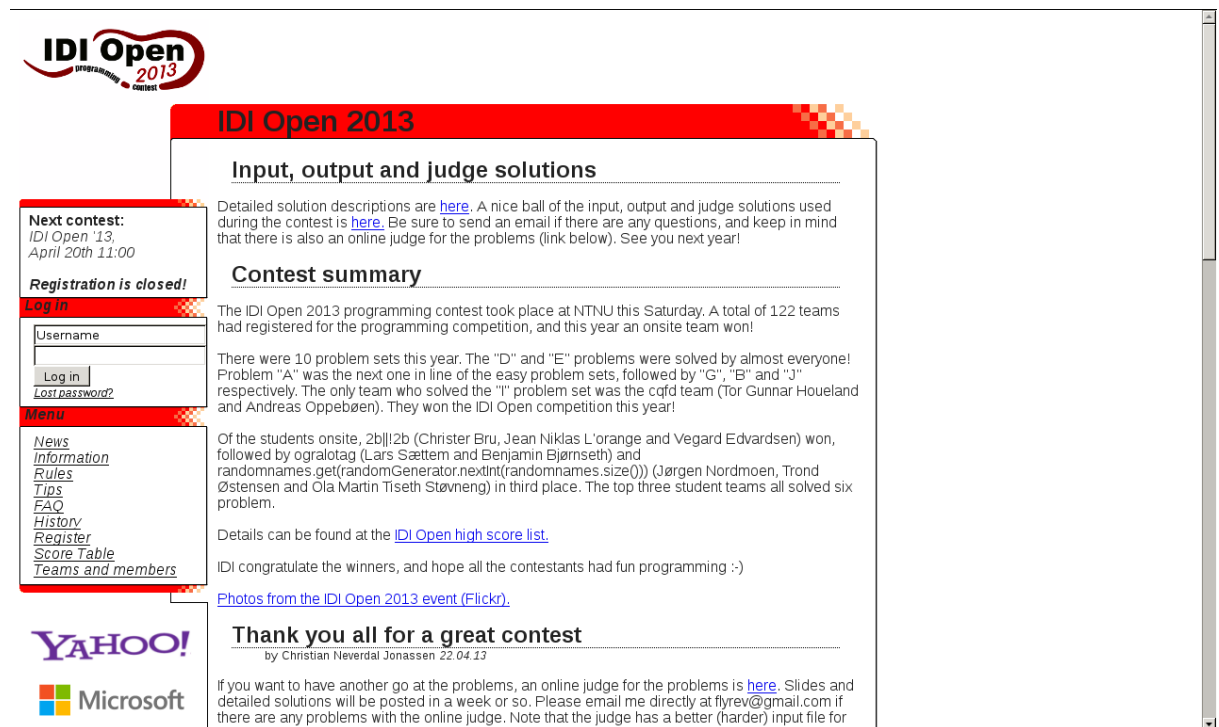


Figure 3.7: FIGURECAPTION

Fig 7.1 We chose to create our own design from scratch, while still trying to keep a similar placement of elements from the previous design. The customer expressed concern regarding how contestants would react to the transition from the old interface to the new one. With this in mind we started to create mockups modelling core elements of the website. Our initial drafts consisted of simple

rearrangements of elements found in the old web interface.

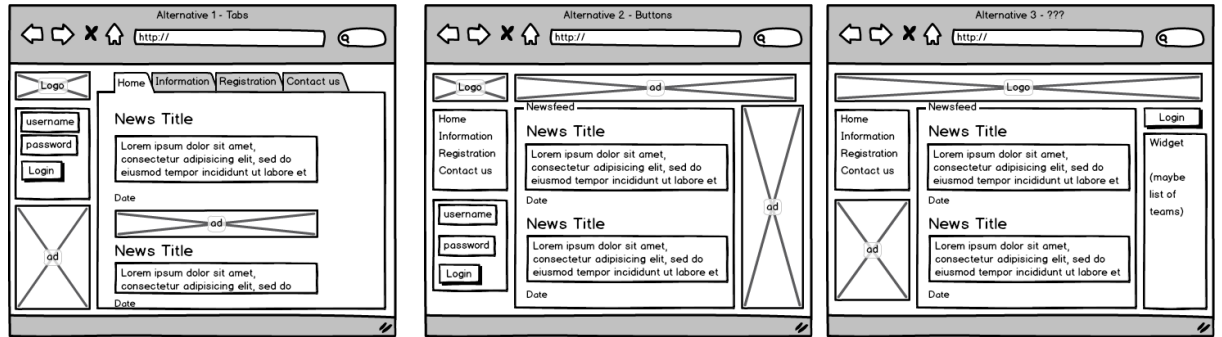


Figure 3.8: FIGURECAPTION

Beyond our three initial mockups we tried a couple of “out of the box” approaches to our designs, but none of them met our standard and was rejected for either being too time-consuming to implement or too far from what our customer wanted. We had a meeting with our customer, where we showed our mockups, and what are thoughts on design had been so far. We wanted to make sure that the customers was on the same page as us, and that we were not moving beyond the scope of the project. Our customers wasn’t very focused on the design aspect, but one demand they had was that they wanted the new site to have the same structure as the old one. One example of what this means is that the customer wanted us to keep the menu on the left-side as you can see that the old system has in Fig 7.1. We agreed, because getting used to a new website can take time, so keeping the structure similar would ease the transition for our users. With this in mind we decided to go for one of our initial mockups, the rightmost one in Fig 7.2, because it had the same structure as the old page, and we personally favoured that design. As a result, most of the elements found in the old interface can be found in the new one, and the transition between using the two is reduced to a minimum.

The task had to be completed in time for milestone M-03, so our main concern was designing for the functionality needed for that particular milestone. However, we also had mockups for functionality outside of this milestone. After milestone M-03 was done, we introduced new design for new functionality through continuous work on top of a template.

The majority of the front end is stylized using bootstrap[Link til kilde] as a framework, enabling us to create a site which is both highly maintainable and aesthetically pleasing at the same time. The admin interface was created using django-admin-interface with Grappelli as a skin to give it a modern look. This worked more or less automatically.

The final page looked like this:

The “black” frame was in our initial page coloured blue, but was changed one week before M-07, idiopen [REMARK: may be altered ].This illustrates

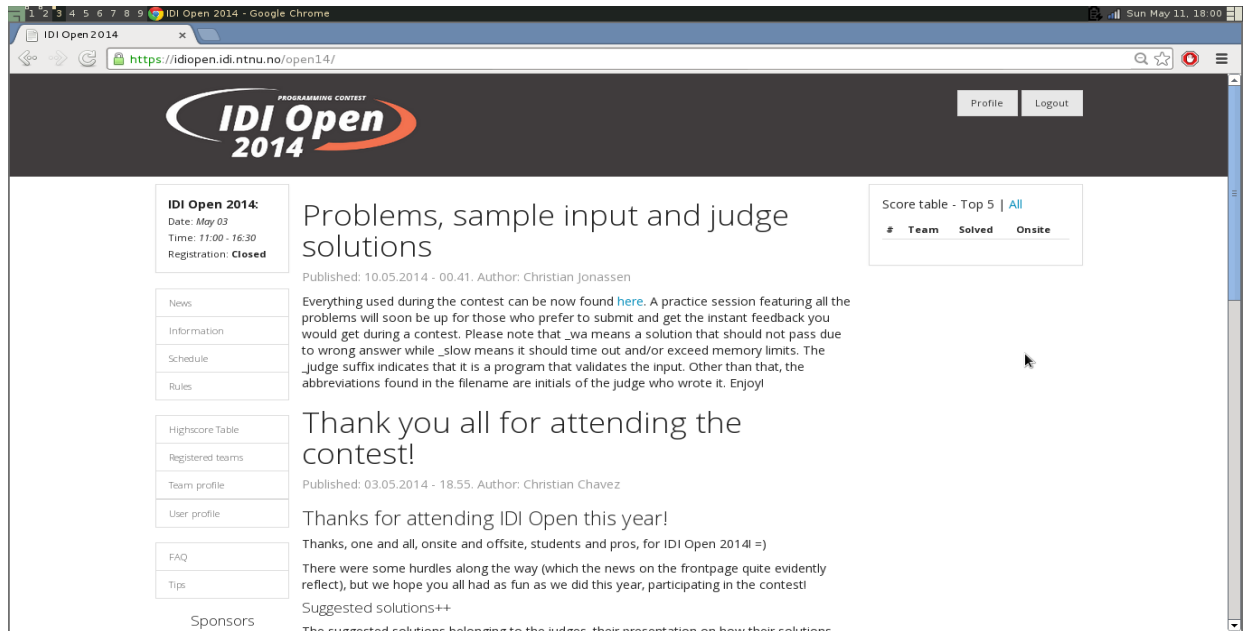


Figure 3.9: FIGURECAPTION

the strongest functionality of the design, namely customization. It is possible, by only uploading a new CSS file, to change the whole feel of the website and give every contest its own theme. The change on IDI OPEN 14, from blue to black, was done as a consequence of a logo change by Richard Eide, one of IDI Open’s facilitators. The old color scheme can be viewed in appendix [insert which appendix]. By comparing fig 7.1 and fig 7.3, you can see that we kept the same structure, but still made some significant changes to the design.

User interface The user interface is designed by using a base template. The template is the same for every part of the webpage, and contains a content block that changes while you navigate through the different parts. This makes it easier to add new content to the user interface, because you already have the base, and don’t need to worry about the header, footer or the menu. We wanted to make it easy for future developers to take over GentleIDI after us, and therefore we focused on a versatile user interface, in case they want to add new functionality.

The menu is placed to the left, coping with the western norm stating that eye placement is natural to the left<sup>1</sup>. We designed the menu to be versatile. Admins can choose what they want to show in the menu, except for *Register user* and *Register team* that are “hardcoded” on request from the customer. This was highly prioritized by our customers, they wanted to be able to make

<sup>1</sup> <http://research.microsoft.com/en-us/um/people/cutrell/chi09-buscher-cutrell-morris-eyetrackingforwebsalience.pdf>

changes without having to change the code. As mentioned in Design process 7.1, we designed the user interface after a principle of versatility. Admins can also change the logo, the sponsor images and the contact information in the footer.

Buttons, images and icons were surrounded with boxes, for example the sponsors and the menu buttons, to show that they are different elements.. There is also one big box surrounding a group of elements, for example the sponsors. This is consistent with the gestalt law of proximity, that constitutes that humans will naturally group objects that are close to each other, and view them as a distinct. This helps the user quickly understand the user interface.

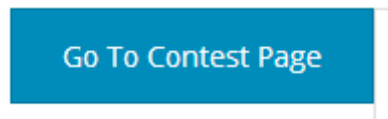


Figure 3.10: FIGURECAPTION

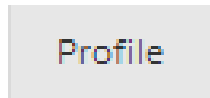


Figure 3.11: FIGURECAPTION

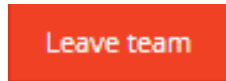


Figure 3.12: FIGURECAPTION

fig 7.4

“To strive for consistency” is the first of Shneiderman’s eight golden rules of interface design<sup>2</sup>, and we tried to follow this while making design decisions. As can be seen in fig 7.4, we decided to use colours that represents the action each button is connected to. The red button marks that pressing this will have permanent consequences. We added a textbox prompt that the user has to answer after pressing a red button, that constitutes to Schneiderman’s fifth and sixth rule, for easy reversal of actions and error handling. This wasn’t added initially, but we noticed while testing the system that without a prompt, it could be possible to leave your team by mistake. fig 7.5

For the contest page, fig 7.5, we wanted to give the contestant a good overview of all the problems, their submissions to them, last feedback, if they solved the problem and the score. It is important to not bury information to

---

<sup>2</sup> <https://www.cs.umd.edu/users/ben/goldenrules.html>



# Contest Page

[Clarification](#) | [Ask a question](#) | [View score table](#) | **Team score: 0**

## List of Problems

Click on a table row to go to the selected problem.

*Hover over each title in the table to get a further explanation.*

Problem ▲	Last Submission ⚡	Time ⚡	Feedback ⚡	Solved ⚡	Score ⚡
Abandon Ship [PRACTICE]					

Figure 3.13: FIGURECAPTION

deep in a website. It could be challenging to balance this while trying not to overload the page with too much information. We had this in mind when designing this page. We got valuable feedback from the customer concerning what they wanted to be present on the contest page. They wanted it to be easy for the contestants to access everything they need, during the competition, through the contest page. After feedback from the customer, we added links to the clarification page and highscore table on the contest page. This lowers the short-term memory load on the contestants, which is consistent with Shneiderman’s eight rule, because they will have everything accessible on the same page.

### Admin interface

The admin interface is developed as an extension Django’s admin interface. Django comes with an extensive admin interface, that provides functionality for adding, removing and changing parts of the system. The admin interface consists of everything we as developers want the admins to be able to change. For a complete listing, see figure X.X[kap 2]. We decided to use Grappelli, an app for the django admin interface that also provided us with more adequate functionality, e.g. auto-completion, rich text editors, drag ‘n drop and more.

The structure of the layout is simple. Each category has it’s own header and everything in blue is clickable. The “Recent Actions” box is there to help admins remember what they last did, which is important to reduce the users short-term memory load, in accordance with Shneiderman’s eight rule.

Originally all the names of the elements were the same as our model names. We decided to change this to more intuitively understandable expressions after a request from the customer. Django’s admin interface couldn’t give us all the functionality we wanted, so we had to extend the interface with our own custom views. We created two views, “Balloon overview” and “Judge overview”. To avoid having to create a similar interface as the rest of admin site, just with

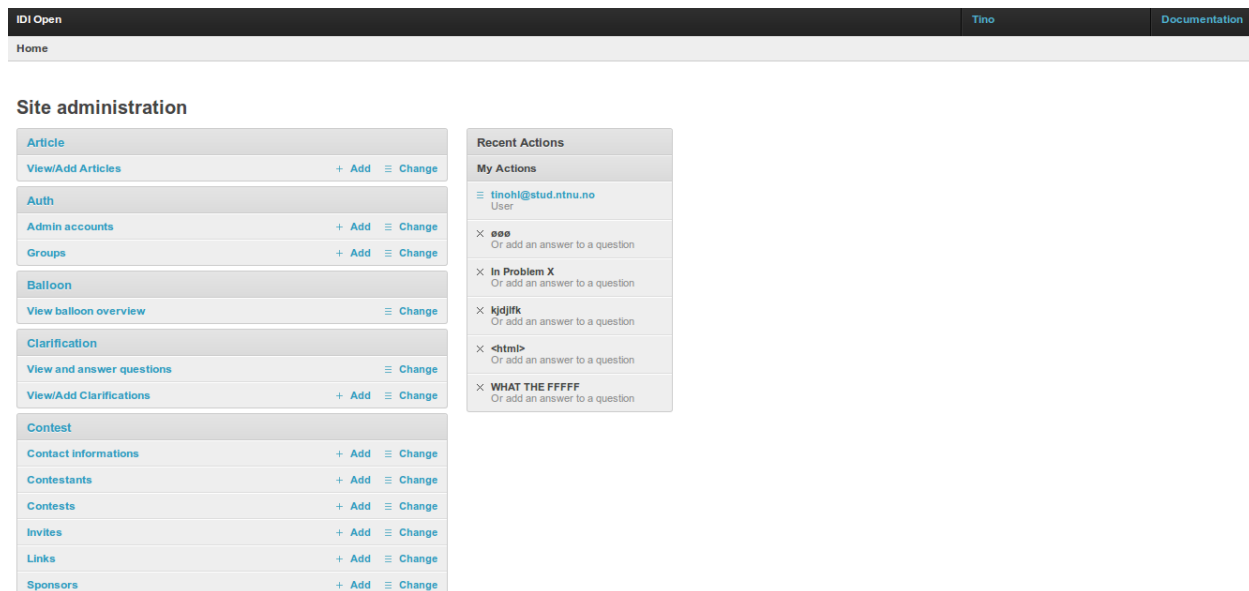


Figure 3.14: FIGURECAPTION

different functionality, we decided to extend the interface templates used for the django admin interface. This allowed us to change what we wanted, while it still kept its consistency with the other parts of the admin site.

fig 7.7

The judge overview was made primarily for judges, but could also be used by the admins. The motivation behind making this view, is that it gives the judges an easier overlook over the competition and how the progress is going for the different teams. We were initially told that the judges wanted a way to see if a team was struggling, so they could help that team.

The view consists of four different tables, with the same layout as the balloon tables. The first two tables depicts how many failed attempts an onsite or offsite team has. The Problem Overview table provides statistics on each problem for the given contest. This was added so that the judges can see which problem has the most failed or successful attempts, and if necessary make changes. To make it easy for the judges to choose a specific team, independent of submissions, we made a dropdown menu with all the teams. The last table is the highscore list. We wanted everything to be on one page for the judges, so they wouldn't have to constantly switch between different pages.

Figure [Judge\_overview for Team]

Figure [Judge\_overview for Team] shows the judge overview after selecting the team "GentleCoding". It is possible to expand each submission by clicking on it. The third submission has been clicked on, so we can now choose to expand different categories. For example if a judge wants to see the source code for that

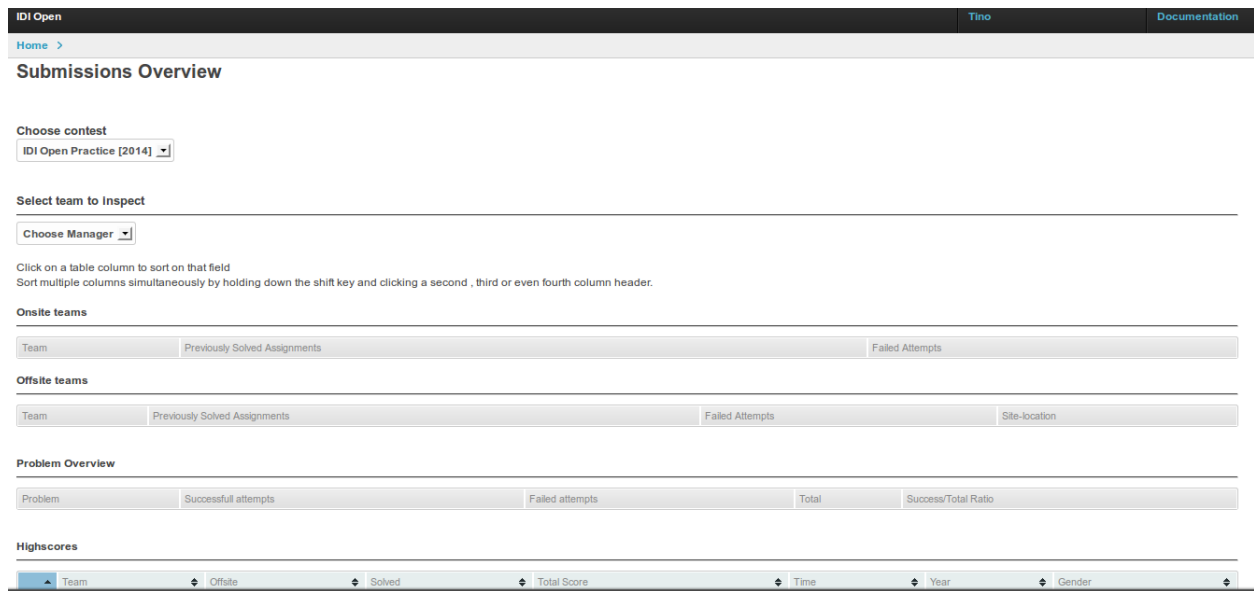


Figure 3.15: FIGURECAPTION

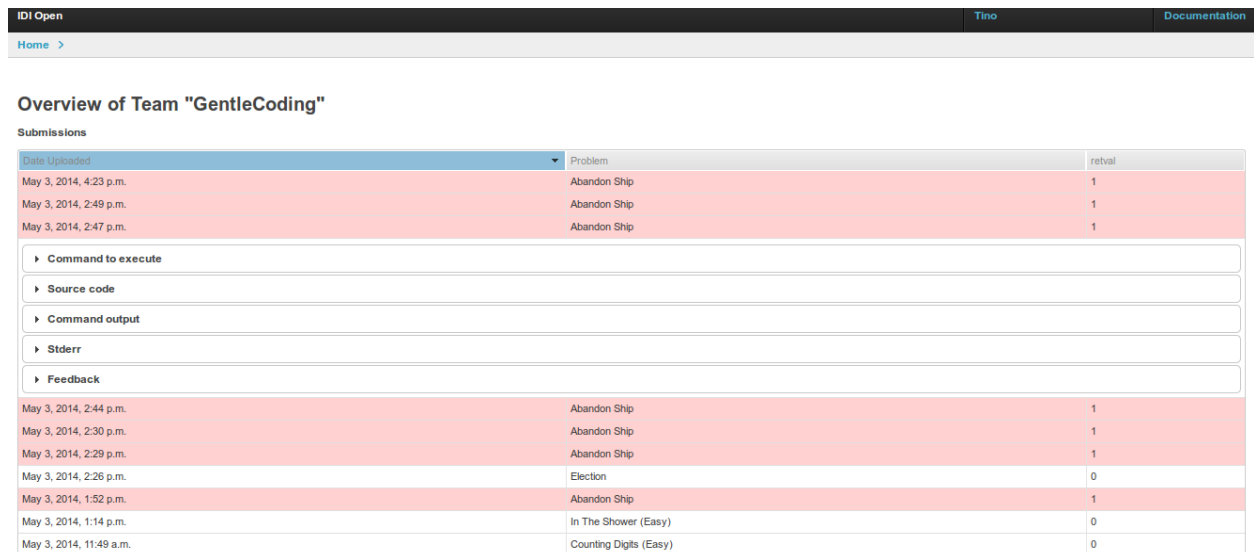


Figure 3.16: FIGURECAPTION

submission, he/she can click on “Source code” and it will expand. Submissions that haven’t been compiled are shown in red, and the other are white.

<https://www.cs.umd.edu/users/ben/goldenrules.html>

## Chapter 4

# Implementation

This chapter goes into the details of our implementation. As mentioned in section X.X, Django follows the MVC pattern in a quite strict manner, and as a consequence so does our project. In addition to MVC our project is divided into several django apps, which are separate modules containing their own models, views and controllers. The apps are intended to serve a specific purpose and provide a certain level of modularity. However, some apps are dependent on others.

Figure X.X shows the directory structure to one of our apps, all apps follow this structure. An app's root folder contains four files worth taking a closer look at, `models.py`, `views.py`, `forms.py`, and `admin.py`.

- `models.py` contains the apps models i.e. our database entities. Due to our site being MVC, every aspect of the site is in some way represented by a model defined in a `models.py` file.
- The file `views.py` defines the app's functions for handling requests, called views. Though the naming might be confusing, the views defined in this file are not views in the MVC sense of the word. The views are in essence MVC controllers. When an HTTP request is received by Django it is routed to a specific view, and the view then handles the request. Though most views simply serve web pages in response to get requests, there are no limits as to what a view can be used for.
- The `forms.py` file contains a set of django forms, which are simply collections of input fields. The forms can be rendered as HTML, and serve as a validator of the input received by POSTs.
- This leaves the `admin.py` file. Django provides a quite modular and modifiable admin app for managing other apps. The admin page's main functionality is that of viewing, editing, creating and deleting models. However, the admin app does not have access to all of the models in the system

# Contest app

```
.
├── admin.py
├── forms.py
├── __init__.py
├── migrations
│   ├── 0001_initial.py
│   ├── 0002_auto_chg_field_team_name.py
│   ├── 0003_auto_add_field_contest_penalty_constant.py
│   └── __init__.py
├── models.py
├── templates
│   ├── Cage
│   │   └── cage.html
│   ├── contest
│   │   ├── alreadyContestant.html
│   │   ├── editTeam.html
│   │   ├── index.html
│   │   └── team.html
│   ├── registerForContest
│   │   ├── registrationComplete.html
│   │   ├── registration.html
│   │   └── requireLogin.html
│   └── viewTeams
│       └── viewTeams.html
├── templatetags
│   ├── __init__.py
│   ├── link_tags.py
│   └── widget_tweaks.py
├── tests.py
├── urls.py
└── views.py
```

Figure 4.1: App overview

by default. The `admin.py` file is where an app registers which of its models are to be modifiable by the admin page, how the models are to be rendered etc.

- The apps also contain a templates directory. The templates are in essence html files that can be easily processed/modified by django. When a view sends a response it is usually by inserting dynamic content(models) into a template and then serving the final html file as an HTTP response. Though not visible in the figure, most of our templates are extensions of a global base template, this way redundancy is reduced and our user interface stays consistent.

## 4.1 contest

The contest app contains the most fundamental functionality and models, the ones related to creating and deleting contests. Just about every other model in the project is related to the contest model in some way. Other than the contest model the contest app defines a couple of models for storing information directly related to a contest, such as sponsor information, support contact information etc.

A complete overview of the models defined in the contest app can be found in [reference contest ER]

## 4.2 article

The article app provides basic functionality for posting news articles. It contains several different views for looking at articles, lists of articles etc. For editing articles the app uses a WYSIWYG editor, available in the admin interface.

## 4.3 userregistration

As the name suggests this app handles user creation, deletion and modification. The majority of this app is an open source app that we incorporated into our project, however, we made some slight modifications.

## 4.4 teamsubmission

When a team has reached something they think might be a valid solution to a problem they submit their source code to the system. The uploaded source becomes part of a submission model which is part of the teamsubmission app.

## 4.5 execution

The system needs a way of handling the submitted source code. For instance it needs some way of determining which compiler is to be used. When the source has been built the system needs to know what command is to be issued to the system to execute the binary. Both of these things are handled by the execution app. In addition there are restrictions set to limit the resources available to the submissions, for example the number of subprocesses, memory allocated etc.

The models defined in `teamssubmission` and `execution` can be found in the [reference submission ER]

## 4.6 node\_manage

With a well configured system and the previously mentioned apps working properly, a submitted source file will be stored and the outline of how the file should be treated will be set when the file is uploaded. The code for actually performing the actions of building and running is handled by the `node_manage` app. The `node_manage` app fetches the appropriate settings for a submission, and submits it to a FIFO queue. Our backend consists of several execution nodes connected in a cluster powered by a framework called `celery`. The nodes can be configured to handle any number of concurrent submissions, and when a node has got available capacity it fetches another submission from the queue. `Celery` relies on the AMQP message passing standard by means of an open source message broker system called `RabbitMQ`. All messages passed go through a broker setup on the same host as the web server, the broker then distributes the messages to the appropriate host.

## 4.7 balloon

When a team has solved a problem, they are to be awarded a helium balloon. This app enables staff users to view problems that have newly been solved by a team, send somebody to deliver a balloon, and then remove them from the list of newly solved. This app is in other words little more than a custom view in the Django admin page.

## 4.8 changeemail

Since we had to modify the `userregistration` app that we incorporated, not everything worked as we wanted out of the box. An example was that the functionality for changing the email of a contestant broke the team that contestant was partaking in. This app provides a fix for that problem and makes sure that changing email works properly.

## **4.9 judge\_supervise**

This app provides judges with an interface in which they can see all submitted solutions and statistics for each team. For each submission, the judges can see compiler errors, execution output and source code.

## **4.10 clarification**

During a contest questions can be asked by contestants to the staff. If a problem is ambiguously formulated, or they are experiencing system errors, these problems can be addressed. The questions are posted publicly on the website, as well as their replies.



## Chapter 5

# Development

This document describes the different phase of development the group went through in order to finish the product. To increase readability the first part of the document describe the process working towards the milestones, as can be viewed in fig 3.3. The second part will describe each sprint in more detail including work done/completed.

### 5.1 Towards the milestones

#### 5.1.1 Milestone M-01 - Preliminary report.

From start to 09.02

Eager to start, we had our first meeting 15.01.2014. During this meeting we discussed which task we wanted apply to. We decided on “IDI\_ProgrammingContest”, which we received.

After receiving the project assignment, we discussed our ambitions for the course and the end product. We agreed that we had a shared goal to receive a top grade in this course, and we that we would resolve in whatever means necessary to achieve this goal. The group was in doubt if we should try popular, enterprise-level tools and frameworks, or if we should stick to basic, previously used tools. We decided to let each member of the group to explore a tool on his own and present his experience to the others. If the tool seemed usable, we incorporated it into our project.

Our primary concern was that we would spend time on suboptimal tools, methods or frameworks. Thus, the group spent much time discussing and modeling the application to come.

### 5.1.2 Milestone M-02 - Mid-semester report

From 09.02 to 09.03

Being aware of the large amount of programming ahead of us, we aimed to have the mid-semester report finished one week before the actual deadline.

To shorten meeting time and strengthen our task overview, we had a meeting thoroughly discussing how SCRUM worked. We decided to adhere more of the conventional SCRUM standard. As a consequence we started to draft release and backlog. This resulted in a reduction in the number of hours used to administering and task delegations. We also got a better overview of the final application. This meant that we could reduce the amount of models and focus more on the code.

The mid-semester report finished as planned one week before our deadline. We more or less completed our testing plans and concluded on management structure. The biggest challenge was how to implement support for user handling.

### 5.1.3 Milestone M-03 - First release

From 09.02 to 19.03

Having finished the midterm report, the group now had a structured overview of the requirements specification and approach for development. We had much coding to do in order to reach the third milestone. We tried to agree on an optimal approach, but concluded that we had to “just get started”. In our sprint backlogs the amount of coding assignments grew. To induce more coding, we arranged informal coding nights in order to trigger “learning by doing” and improved our progression.

By the time we had finished the necessary pre-studies and requirements we already had some functionality. However, there were still work remaining, as suggested by our work breakdown structure. In addition we had a meeting with the customer where they proposed some new requirements, and reprioritized a few other.

In advance to the first release we had some meetings with the customer. We were a little nervous regarding some of the design choices. The meeting discussing the design went well. We had formerly agreed on our mock up-design. There were however a few discrepancies between the delivery and what the customer wanted.

Our first deadline with the customer 19.03, but the actual release of the website was delayed to after the weekend, for external reasons.

#### **5.1.4 Milestone M.04 - Presentation**

From 09.02 to 19.03

Since the presentation was scheduled at the same time as our first release we did not have a time to prepare much for this presentations. Nevertheless, we received valuable feedback from other groups.

#### **5.1.5 Milestone M-05 - Beta Release**

From 19.03 to 11.04

Working toward the beta release was challenging. Increasingly, we experienced that modeling the application before coding was not an optimal solution. Thus, we began to code without relying on diagrams to aid us. We sustained this approach until the end of the project.

With limited time, it became necessary to prioritize some tasks over others. Our improved product backlog, created earlier, proved to be a big benefit. As mentioned previously, we felt that it was hard to predict the outcome of the development process, so we decided not to update the gantt-diagram. Instead we relied on our own options and customer prioritizations. This was due to our new understanding of what needed to be completed when.

We did make some progress with our development, but still had some aspects of our frameworks that needed to be learned. As the weeks went by we increased our work estimates and grew more familiar with the framework. Still our models seldom related to the actual end result. It was not something we felt was a big problem, we did make progress.

#### **5.1.6 Milestone M-06 - IDIOpen test event**

From 11.04 to 26.04

We still had quite a few packages to implement and we were uncertain about how long time we had to spend on each of them. As a consequence we had to shorten our easter vacation. Spending a lot of time together, every day for weeks, may cause tension in groups. We felt it was important to create an environment to ease the tensions. Therefore we took breaks from the coding, eating pizza and playing foosball.

We started every day discussing what we were suppose to do, similar to a daily scrum. We believed all members had a good tacit understanding of what needed to be done, so we transitioned from sprint backlogs to daily TODO lists. These lists were written informally for the sake of brevity.

The days were long, lasting from 09:00 to 24:00. Packages were implemented at

a high pace and we started to become aware of the final product. The biggest challenges were to get the execution node up and running, highscore table and content management for the judges. Testing was also completed. We also had sufficient time to implement some of the lower prioritized requirements.

During the test event, we sat at our own table and received feedback from the judges and volunteers that had shown up. The fact that some of the judges were considered really good programmers made us a little nervous. They did give us feedback and a list of new requirements to be implemented. These were minor fixes, mostly related to the user interface. The test event itself was considered a success: all the judges approved our system.

### 5.1.7 Milestone M-07 - IDI Open

From 26.04 to 03.05

After the test event we got a new list of requirements. There was only one week to the actual event and we had to carefully pick those we and the customer felt were the most important. We implemented support for several execution nodes, refined the content management and fixed small bugs. Some tasks were complex, so it was a challenge to analyze if we would be able to finish them on time. The most advanced tasks we were given after the test event was that the judges wanted a better overview over the contest. More precisely, they wanted access to the whole competition, and all the functionality, before the contest started. The customer also wanted to be able to export data to CSV and LaTeX. This task did, at first, seem lightweight, but turned out to be much more extensive. While finishing on time, this consumed more hours than initially planned.

In total there were 92 teams taking part in IDIOpen14, and a total of 214 registered users in the system. When the contest officially started and the problem sets were released. All users simultaneously accessed the same resource caused a spike on the system load. We were been told by our customer that the old system had previously buckled under the pressure from the spike at the beginning of the contest. Our system did, however, handle this well. Thus, the start of the contest went well.

At one point the system went down for a few minutes. This was because we ran out of hard disk space on our main server. In other words, the system had nowhere to store its data, and was unable to handle the requests made by users. After a couple of minutes of deleting unnecessary files we discovered that for every file that we removed we only bought ourselves a couple of more minutes of uptime. Somewhere in the file system there was a file growing at an alarming pace. To identify this file was challenge. By monitoring the server's processes we found that the application responsible for load-balancing was logging every event under "debug" setting. This meant that every request to the webserver

caused an extensive log entry. This resulted in a 1MB/S disk write rate. The rate was small enough so that we could easily monitor and periodically erase the logs to clear out disk space. We could have disabled logging, however, that would have required a restart of the database server and thereby downtime.

After this problem was resolved the rest of the contest went without any significant issues. At our peak system load we handled a total of 12 concurrent submissions which was more than enough. The website was responsive and working properly. Except the highscore list, which we knew had performance issue. These issues did not have a significant impact for the user experience.

### **5.1.8 Milestone M-08 - Final report**

from 03.05 to 30.05

After the final event we were all exhausted. The following week we only did some administrative tasks. We based ourselves on the mid-semester report and the feedback we got from supervisor and external sources.

Sprint by sprint

We have documented each sprint. These are given in appendix X. An example is given here in table X.X.

# Appendix A

## Sprints

This appendix holds an overview over our sprints, throughout the project. For a more complete list over packages completed see [insert section where activity/sprint backlog are]

This is just an overview where we are trying to bring out the more important aspects of our sprints.

### A.1 Template

Sprint: <sprint nr>	Working towards: <insert milestone>
Overview over packages to be completed: <Insert packages to be completed>	
Improvements: <insert list over things we want to improve about ourself>	
Notes: <any notes>	
Packages completed: <insert packages actually completed>	
Summary: <A brief summary over the most important aspects>	

## A.2 Sprint 0

Sprint: 0	Working towards: M-01
Overview over packages/tasks to be completed: <ul style="list-style-type: none"><li>• Get an overview over the course</li><li>• Get to know the old system</li></ul>	
Improvements:	
Notes: <ul style="list-style-type: none"><li>• This was the first meeting after getting the assignment</li></ul>	
Packages completed:	
Summary: <p>This was still early in the process so most of the time was spent getting an overview over the whole thing.</p>	

## A.3 Sprint 1

Sprint: 0-a	Working towards: M-01
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Read and learn the requirement received from the customer</li><li>• Set up tools</li><li>• Project management</li><li>• Learning tools and framework</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• A better meeting structure</li></ul>	
Notes:	
Packages completed: <ul style="list-style-type: none"><li>• Tools for communication was set up</li></ul>	
Summary: <p>Learning to know the requirements and the subject as a whole was our main concern at this stage. We also did some research on what framework we should use.</p>	



## A.4 Sprint 1

Sprint: 1	Working towards: M-01
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Project management</li><li>• Install and learn tools</li><li>• Report</li></ul>	
Improvements:	
Notes: <ul style="list-style-type: none"><li>• Tino and Eirik was sent out on seminar. Learning about SCRUM</li><li>• Trying to use ICEScrum for Scrum related activites</li></ul>	
Packages completed: <ul style="list-style-type: none"><li>• WBS</li><li>• Risk assignment</li><li>• Functional requirements</li><li>• Class diagram</li></ul>	
Summary: <p>Most of the tools was set up, we started to some modelling, in order to get a better overview over the system to be implemented. This was also documentations to be used in the report. We also systematized the requirements in order to communicate with the customer. Project roles was also distributed.</p>	

## A.5 Sprint 2

Sprint: 2	Working towards: M-01
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Project management</li></ul>	
Improvements:	
Notes:	
Packages completed: <ul style="list-style-type: none"><li>• Requirement specification</li><li>• System architecture<ul style="list-style-type: none"><li>– Flow charts</li><li>– class diagrams</li></ul></li><li>• ER-Models</li><li>• Preliminary report</li></ul>	
Summary: <p>At this point we had a rough understanding of the work ahead of us, and we were able to start modelling possible solutions. This was also close to the deadline for the preliminary report and as a consequence a lot of time was spent on the report.</p>	

## A.6 Sprint 3

Sprint: 3	Working towards: M-02
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Development</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• Better sprint planning</li><li>• We should improve our task delegation</li><li>• We should prioritize tasks</li></ul>	
Notes:	
Packages completed: <ul style="list-style-type: none"><li>• Development</li></ul>	
Summary: <p>During the past two sprints we had primarily been planning and doing administrative tasks. This sprint marked the end of that phase. We moved on to actual implementing. However, we ere not familiar with the tools and frameworks available to us, and as a consequence we decided to use this sprint to get everyone up to date on Django//python. We had a coding night this sprint. Working all members together.</p>	

## A.7 Sprint 4

Sprint: 4	Working towards: M-02
Overview over packages to be completed: <ul style="list-style-type: none"><li>• User-interface</li><li>• Project management</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• The activity diagrams does not reflect upon our actual work done.</li></ul>	
Notes:	
Packages completed: <ul style="list-style-type: none"><li>• User interface</li></ul>	
Summary: <p>During sprint 4 we knew we had to improve our WBS. We had a long meeting where we rebuild our backlog, reviewed SCRUM and created a release- and backlog.</p>	

## A.8 Sprint 5

Sprint: 5	Working towards: <insert milestone
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Development</li><li>• Report</li><li>• Tesplan</li></ul>	
Improvements:	
Notes: <ul style="list-style-type: none"><li>• This sprint we had a meeting with the supervisor discussing the activity diagrams. Show suggested that we switch them with our</li></ul>	
Packages completed: <ul style="list-style-type: none"><li>• Sponsor support</li><li>• Testplan</li></ul>	
Summary: <p>We had a good overview over what should be in the report at this point. A finished version was right around the corner. In general, this weeks meeting went much faster than the last. The group was happy about that.</p>	

## A.9 Sprint 6

Sprint: 6	Working towards: M-02/M-03
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Mid-term report</li></ul>	
Improvements:	
Notes:	
Packages completed: <ul style="list-style-type: none"><li>• Mid-term report</li><li>• Testplan</li><li>• User-interface completed in bootstrap</li></ul>	
Summary: <p>This sprint we finished the mid-term report and the user-interface was completed. We was happy with the resut. We also finished the mid-term in food time before the actual deliver.</p>	

## A.10 Sprint 7

Sprint: 7	Working towards: M-03/M-04
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Representation</li><li>• Implementation</li><li>• Write tests.</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• We must be better to work with other group members code.</li></ul>	
Notes:	
Packages completed: <ul style="list-style-type: none"><li>• Login completed</li><li>• User registration completed</li><li>• Team registration completed</li></ul>	
Summary: <p>During this sprint we had boost with the implementation. We were busy making our Firs release. Unfortunately we did not have time to set up the solution live this sprint .It was postponed to after the weekend.</p>	

## A.11 Sprint 8

Sprint: 8	Working towards: M-05
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Testing</li><li>• Set up solution live</li><li>• Fixing bugs</li><li>• Peer evalutaion</li></ul>	
Improvements:	
Notes: <any notes>	
Packages completed: <ul style="list-style-type: none"><li>• Testing</li><li>• Bug fixing<ul style="list-style-type: none"><li>– Change email</li><li>– Forgot password</li></ul></li><li>• Peer evalutaion</li></ul>	
Summary: After we put the solution up, there was sum bugs and testing to be done. We had not had the opportunity to test, by our standards, yet. We did this while the solution was live.	



## A.12 Sprint 9

Sprint: 9	Working towards: M-06
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Implementation</li><li>• Permission testing</li><li>• user manual</li><li>• Project mamagement</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• We had to be more consistent with testing</li><li>• Better to fill out sprint documents.</li></ul>	
Notes: <ul style="list-style-type: none"><li>• We received the Peer Evaluation.</li></ul>	
Packages completed: <ul style="list-style-type: none"><li>• Possible to upload solutions</li><li>• Models</li></ul>	
Summary: <p>This sprint was probably our worst planned sprint. With better planning we could have finished a lot more coding. Unfortunately this was not the case and we spent unnecessary much time in the wrong direction. We were, however happy with our peer evaliation.</p>	

## A.13 Sprint 10

Sprint: 10	Working towards: M-05
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Implementation</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• Still improvement to be done with filling out sprint backlog.</li></ul>	
Notes: <ul style="list-style-type: none"><li>• This sprint was 9 days long</li></ul>	
Packages completed: <ul style="list-style-type: none"><li>• Implementation<ul style="list-style-type: none"><li>– Execution nodes</li><li>– Compiler profiles</li><li>– Upload solution</li></ul></li></ul>	
Summary: <p>This was the last sprint before Easter. We were more thrilled with this sprint but, we knew had to shorten our easter vacation. We had a good start with much of the implementation and we finally felt like we had a good overview over everything.</p>	

## A.14 Sprint 11

Sprint: 11	Working towards: M-05
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Implementation</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• We knew we needed discipline to make it</li></ul>	
Notes: <ul style="list-style-type: none"><li>• Parts of this sprint was during easter</li><li>• This sprint was 11 days</li></ul>	
Packages completed: <ul style="list-style-type: none"><li>• Upload submission</li><li>• Penalty systematized</li><li>• Review system status</li><li>• Judge supervisor</li><li>• Error messages</li></ul>	
Summary: <p>During this sprint, we did not setup a sprint backlog. Instead we kept an well documented TODO list. Every day all members would tell which tasks from the TODO list they would work on. At the end of the day we told each other what was missing. This sprint went great and we were actually finished some days before M-05-.</p>	

## A.15 Sprint 12

Sprint: 12	Working towards: M-07
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Development</li><li>• Bugfixes</li><li>• Setup</li></ul>	
Improvements:	
Notes: <ul style="list-style-type: none"><li>• Last sprint before final event</li></ul>	
Packages completed: <ul style="list-style-type: none"><li>• Highsvore</li><li>• CSV and PDF support</li><li>• Several execution nodes</li><li>• judge contest access</li></ul>	
Summary: <p>Are last sprint before the final event consisted mainly on small bugfixes. There were, however, some tasks that took longer time than estimated. That would be CSV and PDF</p>	

## A.16 Sprint After

Sprint: After	Working towards: M-08
Overview over packages to be completed: <ul style="list-style-type: none"><li>• Final report</li><li>• Small bugfixes</li><li>• User Manual</li></ul>	
Improvements: <ul style="list-style-type: none"><li>• Efficiency and communication is import this last period</li></ul>	
Notes: <ul style="list-style-type: none"><li>• We did create a traditional sprint backlog for this sprint. We did however have frequent meeting discussing what to finish when</li></ul>	
Packages completed: <ul style="list-style-type: none"><li>• Final report</li><li>• small bugfixes</li><li>• User manual</li></ul>	
Summary: <p>When we worked towards the final report we decided on a different tactic than the other sprints. Instead of creating a sprint backlog, holding all the tasks, we broke down the report into chapters. Some of which was already finished. For each chapter we talked about what key points we wanted to write about for so deciding a pair that should write that part. Then, before we met next time, another pair would view, comments and generally share some points about that chapter.</p>	

Sprint: After	Working towards: M-08
---------------	-----------------------

## Appendix B

# User stories

Admin

ID:	Priority	Story
SA-01	HIGH	Will be able to create a new contest. When doing so a new web page should be created, but whether the site should be immediately published or not is optional. The content of the new site follows a strict template, but adding a custom css-file will be possible. Each contest has got its own settings, containing a list of supported compiler profiles, compiler flags, penalty system, maximum number of contestants, maximum number of contestants per team, and of course a date and a name. When creating a contest the admin needs to provide a name and a date, the other settings may be skipped and default settings will be used.
SA-02	HIGH	Users are organized in user groups(admin being one of them). By default three usergroups are provided, admin, judge, contestant and functionary. The entire solution is based on independent modules of functionality and each user group has got access to a subset of these modules. The admin is the only non-modifiable user group, admins have access to all modules. The admins can modify all other user groups, change permissions of a group and remove/add member to a group, this includes promoting new admins. The admins are also able to deactivate users, and even remove them from the database.

SA-03	MED	The system is able to gather a large variety of statistics, what data is to be collected is decided by the admins.
SA-04	HIGH	The system uses a collection of nodes(computers) for assessing submissions. The admins can add a node by providing an IP address and the username and password of a privileged user on that node. These nodes can also be removed by the admins. The nodes can also be managed in terms of compiler profile support.
SA-05	HIGH	The web page associated with a contest consists of a set of news items, these can be added by the admin. As with the entire contest web page the publishing of the news item can be set to a certain date and time. The news items can also be removed or modified later on.

Role: Judge

ID:	Priority	Story
SJ-01	MED	A judge can submit a problem, where he/she will be able to upload cases with input/output. He/she can give every case a name. For each problem the judge can set a resource limit (time + memory) for each compiler profiles. He/she can upload different solutions that gives the right output, timeout and the wrong answer. All the solutions should be run-able and produce an output about the expected result, and if the execution time is inside the given boundaries. He/she should also be able to check that all problems have associated solutions that give right and wrong answer, and timeout.
SJ-02	MED	A clarification system will be available to judges, where they can receive and respond to messages from contestants. When receiving a message, the judge will get a notification (possible in in the bottom right corner of the website, [Design choice]) . A judge can choose to either send a global message or a message to a contestant or a team. A global message will be sent to every contestant in the competition.

Role: Contestant

ID:	Priority	Story
SC-01	HIGH	A contestant should be registered with an email, name, gender, and study programme and level. When registered, he/she should receive a confirmation email. After confirming the account, a contestant should be able to log in.



SC-02	HIGH	When a contestant is logged in he/she will have access to account information and which teams he/she are invited to, as well as earlier contests and teams they have participated in. The contestant should be able to edit account information
SC-03	MED	A clarification system will be available to contestants, where they can ask questions to the judges. They will also have access to answers the judges have marked as global.

Role: Functionary

ID:	Priority	Story
SF-01	LOW	When a team completes a problem, a table containing the group name and location should be updated to include this. Each problem has a corresponding balloon colour. A balloon functionary should be able to register a balloon colour to each problem.

Role: Teams

ID:	Priority	Story
ST-01	HIGH	A contestant must [18.02] be able to register a team, upon registration he/she is required to input team name, whether or not the team is onsite, a team password, and a email for the team leader.
ST-02	HIGH	The team leader should be able to edit the team information, invite new members, and delete the team before the competition. To invite new members you input their email, and they receive a registration link, where he/she inputs name, gender and nickname. If the contestant [changed from email 20.02] is already in the database from a previous competition, the email they receive contains a confirmation link. Every contestant can manage the team they are a member of. All informations is editable in the team overview which can be reached from a contestants login. A confirmation email is sent to the edited user.

ST-03	MED	A team should be able to deliver submissions to problems, and get a response from the system. The response should be whether the submission is right, wrong, or gives timeout.
-------	-----	--

## Appendix C

# Installation guide

This is the complete install guide for GentleIDI. The guide will assume that the reader has got some basic linux skills. You should be capable of installing packages by means of a package-manager like apt, yum etc.

Though GentleIDI is not tightly linked with any specific linux distro, this guide assumes that you're using Ubuntu Server 14.04. This is the only distro on which the system has been tested thoroughly at the time of writing.

GentleIDI is in many ways a straightforward django-based website, and hence there are a lot of possible setups to choose from. This guide is inspired by a guide written by Michal Karzyński, and will guide you through the steps of setting up the system using a combination of gunicorn and nginx.

### C.1 Creating your users

Running a website as a user with root privileges or anything of the sort is far from recommended. Therefore you are recommended to create a new user and a new usergroup. The names of both the group and the user can be chosen as you please, but the rest of the guide will stick to using a user called gentleidi and a group named webapps.

```
$ sudo mkdir -p /webapps/gentleidi
$ sudo groupadd --system webapps.
$ sudo useradd --system --gid webapps --home /webapps/gentleidi gentleidi
$ sudo chown gentleidi:webapps /webapps/gentleidi/
```

Now you have a user named gentleidi which is a member of the usergroup webapps, and whose home directory is /webapps/gentleidi.

In addition to the user we just created, we need another user, specifically used to run the untrusted software submitted by the contestants. GentleIDI assumes that this user is named gentlemember. However, changing this value in the source is no complicated matter.

```
$sudo useradd --system gentlemember
```

The system needs to be able to execute commands both as gentleidi and gentlemember. As the web server runs as gentleidi we need to make sure that gentleidi can execute commands as gentlemember. Add the following line to your sudoers file.

```
gentleidi ALL=(gentlemember) NOPASSWD:ALL
```

If you don't know how to edit your sudoers, to open the sudoers file in a text editor simply type the following command:

```
$sudo visudo
```

Now we've got two users, one capable of executing commands as the other. What we want to do now is to ensure that gentlemember is unable to communicate via network. This is done by applying two rather straightforward iptable rules.

```
$ sudo iptables -A OUTPUT -m owner --uid-owner gentlemember -j LOG
$ sudo iptables -A OUTPUT -m owner --uid-owner gentlemember -j REJECT
```

Though this will restrict the user's network access, be aware of software installed on your system which is capable of switching to another user.

## C.2 Setting up the environment

Due to a lot of strict changes made in python versions, a lot of libraries do not work across different versions of python. This leaves python in a situation where program A might need python to be version X and program B might need python to be version Y. So, what do you do? You setup a virtual environment.

Virtual environments is a way of setting up separate python setups for different sets of programs.

What we want to do is to turn the home directory of the gentleidi user into a virtual environment.

```
$ sudo apt-get install python-virtualenv
$ sudo su gentleidi
$ virtualenv ~/env
```

Now that you've got a virtual environment you can start filling it with something useful, like the content of the project's git repo.

```
$ cp -r /path/to/repo/IDIOpen/ /webapps/gentleidi/
```

Please note that you only need the wsgi folder from the repo, however, updating is a lot easier when all you've got to do is pull the latest version directly using git. The downside is that you could possibly end up committing your production system configuration files etc to the repo. However, we're going to assume that you will not be developing directly in your production system, and thereby avoid the hazard.

Before leaving this step, ensure that the files in /webapps/gentleidi has got the correct file permissions.

```
$ sudo chown -R gentleidi:webapps /webapps/gentleidi
```

## C.3 Installing required packages

Now it's time to start making sure that you've got the packages you need to run GentleIDI.

```
$ sudo apt-get install git nginx libmysqlclient-dev python-dev
```

You might already have most of these packages, however, better safe than sorry.

The next thing you need to do before continuing is to log in as gentleidi and activate your newly created virtual environment.

```
$ sudo su gentleidi
$ source ~/env/bin/activate
```

Installing the required python packages via pip is easily done. In the project root directory there's a file named requirements.txt. This file is simply a list of required packages, to install them simply execute the following:

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

## C.4 Database

GentleIDI needs a database to store its data. This guide will show you how to setup GentleIDI with a mysql database server, however, if you feel like using mariaDB, postgresql, or even SQLite, then please do. Any database server supported by Django is supported by GentleIDI.

Naturally you don't need to install the database server on the same host as the web server, that's what we'll do for now.

```
$ sudo apt-get install mysql-server
```

Now what we need to do is to create a database and a mysql user that GentleIDI can use. During the install process you were required to set a root password for the mysql-server. Login as root and perform the following commands:

```
> CREATE USER 'gentledb'@'localhost' IDENTIFIED BY 'password';
> GRANT ALL PRIVILEGES ON * . * TO newuser@localhost;
> FLUSH PRIVILEGES;
> CREATE DATABASE gentleidi CHARACTER SET utf8 COLLATE
utf8_general_ci;
```

Remember to replace 'gentledb' and 'password' with a suitable username and password.

Now you need to ensure that GentleIDI uses your newly created database. Edit the DATABASES entry in IDIOpen/wsgi/openshift/settings.py

```
if MYSQL:
130     DATABASES = {
131         default: {
132             ENGINE : django.db.backends.mysql,
133             NAME : gentleidi,
134             USER : gentledb,
135             PASSWORD : password,
136             HOST : localhost,
137             PORT : 3306,
138         }
139     }
```

In order to make sure that the database is working properly, login as gentleidi, activate your environment and synchronize GentleIDI's database.

```
$ sudo su gentleidi
$ source ~/env/bin/activate
$ python ~/IDIOpen/wsgi/manage.py syncdb
$ python ~/IDIOpen/wsgi/manage.py migrate
```

If this command terminates properly, then your database should be good to go. In fact you should be able to run GentleIDI on a development server at this point. But first, you need to create a admin account. To do so, simply execute the following:

```
$ python ~/IDIOpen/wsgi/openshift/manage.py createsuperuser
```

To start the development server run:

```
$ python ~/IDIOpen/wsgi/openshift/manage.py runserver
```

You should now have a working website running on port 8000. However, you have no execution nodes available to evaluate submissions, and you're using Django's development server, which scales horribly.

## C.5 Gunicorn

Now it's time to install replace the Django development server with a proper application server, gunicorn. Member to be logged in as gentleidi, and to activate your environment before doing this.

```
$ pip install gunicorn
```

Now we need a script that launches gunicorn and GentleIDI appropriately.  
#!/bin/bash

```
NAME=GentleIDI                                # Name of the application
DJANGODIR=/webapps/gentleidi/IDIOpen/wsgi/      # Django project
directory
SOCKFILE=/webapps/gentleidi/run/gunicorn.sock  # we will communi-
cate using this unix socket
USER=gentleidi                                # the user to run as
GROUP=webapps                                  # the group to run as
NUM.WORKERS=3                                  # how many worker processes should
Gunicorn spawn
DJANGO_SETTINGS_MODULE=openshift.settings      # which
settings file should Django use
DJANGO_WSGI_MODULE=openshift.wsgi             # WSGI module
name

echo Starting $NAME as whoami

# Activate the virtual environment
cd $DJANGODIR
source /webapps/gentleidi/env/bin/activate
export DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
export PYTHONPATH=$DJANGODIR:$PYTHONPATH

# Create the run directory if it doesnt exist
RUNDIR=$(dirname $SOCKFILE)
```

```

test -d $RUNDIR || mkdir -p $RUNDIR

# Start your Django Unicorn
# Programs meant to be run under supervisor should not daemonize them-
selves (do not use --daemon)
exec /webapps/gentleidi/env/bin/gunicorn ${DJANGO_WSGI_MODULE}:application
\
    --name $NAME \
    --workers $NUM_WORKERS \
    --user=$USER --group=$GROUP \
    --log-level=debug \
    --bind=unix:$SOCKFILE

```

Place the contents of the previous page in the following file: `/webapps/gentleidi/env/bin/gunicorn_start`

Make sure that the script is executable:

```
$ sudo chmod u+x /webapps/gentleidi/env/bin/gunicorn_start
```

#### 1. (a) nginx

As mentioned previously this setup relies on a combination of gunicorn and nginx. At this point gunicorn should be working properly, and it's time to setup nginx.

If you have not already installed nginx, do so now:

```
$ sudo apt-get install nginx
```

Now you need to create an nginx configuration file for gentleidi.

Store the content found below in the following file : `/etc/nginx/sites-available/gentleidi`

```

upstream hello_app_server {
    server unix:/webapps/gentleidi/run/gunicorn.sock fail_timeout=0;
}

```

```

server {

    listen 80;
    server_name example.com;

    client_max_body_size 4G;
}

```



```

access_log /webapps/gentleidi/logs/nginx-access.log;
error_log /webapps/gentleidi/logs/nginx-error.log;
location /static/ {
    alias /webapps/gentleidi/IDIOpen/wsgi/static/;
}

location /media/ {
    alias /webapps/gentleidi/IDIOpen/wsgi/media/;
}

location / {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;

    if (!-f $request_filename) {
        proxy_pass http://hello_app_server;
        break;
    }
}

# Error pages
error_page 500 502 503 504 /500.html;
location = /500.html {
    root /webapps/gentleidi/IDIOpen/wsgi/static/;
}
}

#EOF

```

In this configuration nginx is set to log all accesses and errors. These files need to be created by the following commands:

```

$ sudo su gentleidi
$ mkdir ~/logs
$ touch ~/logs/nginx-access.log
$ touch logs/nginx-error
$ exit

```

All you need to do at this point is to enable the nginx site. This is done simply by creating a symbolic link from the configuration file in sites-available to sites-enabled.

```

$ sudo ln -s /etc/nginx/sites-available/gentleidi /etc/nginx/sites-enabled/
$ sudo rm /etc/nginx/sites-enabled/default
$ sudo service nginx restart

```

You should now have a working website. All that's left is making management a little easier and adding some execution nodes.

## C.6 Supervisor

Supervisor is a utility for defining and managing jobs. In this case we're going to define two jobs, one for managing the website, and another for managing an execution node.

You need to create two files to make this happen:

```
/etc/supervisor/conf.d/gentleidi.conf
[program:gentleidi]
command = /webapps/gentleidi/env/bin/gunicorn_start
user = gentleidi
stdout_logfile = /webapps/gentleidi/logs/gunicorn_supervisor.log
redirect_stderr = true
#EOF

/etc/supervisor/conf.d/celery.conf
[program:celery]
command=/webapps/gentleidi/env/bin/celery worker -A openshift -l info

directory=/webapps/gentleidi/IDIOpen/wsgi
environment=PATH=/webapps/gentleidi/env/bin:$(ENV_PATH)s
user=gentleidi
autostart=true
autorestart=true
redirect_stderr=True
#EOF
```

Create the log files that you've referenced.

```
$ mkdir /webapps/gentleidi/logs/
$ touch /webapps/gentleidi/logs/gunicorn_supervisor.log
```

Read the newly created configuration files.

```
$ sudo supervisorctl reread
$ sudo supervisorctl update
$ sudo supervisorctl restart all
```

## C.7 Multiple execution nodes

The easiest way of setting up multiple execution nodes is to clone the setup on your web server to other machines and then making minor changes.

When setting up multiple execution nodes there are two changes that need to be made. The directory `/webapps/gentleidi/IDIOpen/wsgi/private/submissions` needs to be shared between all the execution nodes. How you decide to make

this happen is up to you. However, sshfs is possibly the easiest solution. Whatever way you decide to mount the directory on your execution nodes, make sure that multiple users are allowed to access it, e.g. the `allow_other` option for sshfs.

You also need to make sure that all your execution nodes have access to the same database. Make sure that the `settings.py` is not set to `localhost`, but rather points to whatever host you decide to use as a database server. Some configuration of your database server might be needed in order for it to accept remote connections. mysql servers need to change the `bind-address` property in the `/etc/mysql/my.cnf` to their actual IP, not `localhost(127.0.0.1)`.

You also need to change the grants for the mysql user in such a way that it is allowed to connect remotely to the database.