Politecnico di Milano
School of Industrial and Information Engineering

Master of Science in Computer Science and Engineering
Department of Electronics, Information and Bioengineering

Terry: a new software to support the Informatics Olympiad

Final thesis in
Software Engineering

Advisor
Prof. Giovanni Agosta

Candidate
William Di Luigi — ID: 864165

Academic Year 2018 – 2019

Contents

Introduction

In this thesis, we will give a brief overview on what the Informatics Olympiad is and explain the reasons it exists. Then, we will explore how the national selection for this olympiad takes place in Italy, looking at each selection phase specifically to understand the issues and limitations we face at each step.

Significant space will be devoted to *CMS*, the most important software that we currently use in the Italian Informatics Olympiads. We will see how CMS helps us in running the Olympiads, and what its limitation are.

After discussing the various phases of the Olympiads where CMS is used, we will shift our attention to an earlier phase: the *district-level* contest. This contest comes usually 5 months before the national finals and is used to select the best Italian students to invite to the final itself.

The district-selection is definitely "easier" than the national final from the point of view of a participant, but from the point of view of contest administrators it is a much more delicate and intricate phase.

We will see in detail how the district-level selection is different from the national final and why CMS turned out to not be the "right tool for the job" for it. Finally, we will introduce *Terry*, the brand-new software that we designed and implemented from scratch. Terry is the result of the careful requirements analysis of the district-level selection, and the precious inputs received by the experience with the CMS software as well as other notable contest platforms.

1 The Informatics Olympiads

Most of us are familiar with the Olympic Games, a series of international sporting events in which thousands of athletes from around the world participate in a variety of competitions. The Olympic Games are held every four years, and they are widely considered to be the world's foremost sports competition with more than 200 nations participating [1].

In a similar way, the International Science Olympiads are a group of worldwide annual competitions in various areas of science such as mathematics, biology, and many more [2]. Most of those Olympiads don't require special software to measure athletes' results, except one: The *International Olympiads in Informatics*.

The International Olympiads in Informatics (or **IOI** for short) is a worldwide competition open to high-school students who excel in the skill of *Algorithmic Problem Solving* [3]. The main difference between the IOI and the Olympic Games is that the former involves a mind-activity rather than a physical one, and, of course, that the latter is much older, having had its first edition in 1896[1] in Athens, Greece, when the first IOI ever held was in 1989 in Pravetz, Bulgaria.

[1]The ancient Olympic Games ran from the 8^{th} century BC to the 4^{th} century AD, but the modern Olympic Games were established in 1896.

1.1 Algorithmic Problem Solving

When we talk about Algorithmic Problem Solving (APS) we are referring to a very specific activity: it's a mind-sport in which athletes of all ages compete by sitting in front of a computer writing programs. The "writing programs" part of it might make APS sound like traditional programming work, but it's actually deeply different. In some contexts, APS is called *competitive programming* and, like any activity pushed to a competitive level, even programming acquires its own distinctive traits.

1.1.1 APS versus traditional programming

A significant difference between traditional programming and APS can be found in the *reasons for doing them*: people usually write software for an economic benefit (e.g. they are employed by a software firm that pays them a salary); this is almost never true for APS since, most of the time, people who participate in this activity have no economic incentive to do so. Athletes usually participate for the glory, for showcasing their skills, and for the honor of representing their country in a worldwide championship.

Another difference is that programmers write software that ultimately, in one form or another, will be used by other people; on the other hand, algorithmic problem solvers have no expectation of their program to ever be used by other people. The main goal is to pass the test cases required by the *contest platform* which they happen to be using: once that is done, the athlete usually won't use the source code again in the future.

These two differences are enough to tell APS apart from other software engineering activities. In fact, most lessons and "best practices" which are of critical importance in the software engineering field become simply superficial, useless and borderline dangerous when applied to APS. For example, it's

universally accepted that, in traditional software engineering, code should be well-commented and documented [4]. This usually includes adopting a verbose coding style, using plenty of comments, and adequately separating the codebase in different files or modules. In APS, following these rules would be dangerous (over-commenting code means losing time, a very limited resource in APS) or outright impossible (separating the codebase in different files is not supported by many contest platforms: most of the times, a single file is required).

Participants in APS generally compete alone. There are some team-based competitions e.g. where each team represents a specific school or university but, even then, the actual coding is done by a single person: the team splits the problem set so that each person works on a separate task [5].

In the industry, on the other hand, working on the same codebase is the norm for teams. Teamwork is not only encouraged, it's non-negotiable. Since writing code is for now a human-only activity, the only way to scale software development is horizontally: by adding more people. Version Control Systems (VCS) were invented specifically to help multiple people work on the same codebase. Using a VCS in algorithmic problem solving, however, is unheard of.

Moreover, in the industry, teams lose and gain new members frequently, so it's critical that new people make the least possible effort to understand an existing codebase and quickly get up and running with it. This again leads to why in traditional software engineering it's a good idea to invest time in documenting and properly splitting the codebase.

1.1.2 The goal, in an APS competition

During an Algorithmic Problem Solving competition, the participants are faced with one or more "problems" (or "tasks") to solve in the limited time of the contest. The number of tasks can go up significantly if the contest is

team-based or in cases where the contest lasts for a long time, e.g. multi-day online contests.

Regardless of how many tasks form a contest, each task requires participants to write a program in order for them to solve it. The program will be tested on some input data for which it will return some result. The criteria for evaluating weather a participant solved a task are usually limited to the following three:

1. The **correctness** of the result produced by the program.
2. The **time** it took for the program to report the result.
3. The amount of **memory** (RAM) used to calculate the result.

These three factors are measured and then used to judge whether the solution proposed should be accepted or rejected (or, where applicable, if it should receive a partial score).

Most of the time, in order to complete a task successfully, the solution is required to implement an algorithm or data structure that produces correct results and that uses an **asymptotically optimal** amount of time and memory.

1.1.3 Asymptotic Complexity

Given a program for which we measured the running time (in seconds) and the memory used (in bits) during some computation, we can define the asymptotic complexity of that program. This proves very useful in some cases, e.g. when preparing new tasks, since we can make it so that a solution with a "better" (i.e. lower) complexity will score higher, or so that a solution with "worse" (i.e. higher) complexity won't fit in the time limit.

Let's denote with $f(n)$ the seconds or bytes required by some solution when the input size is n. We will say that $f(n)$ belongs to the class of functions $\mathcal{O}(f(n))$ if and only if, as the input size approaches infinity, $f(n)$ and $g(n)$ only differ by a multiplicative constant. In other words:

$f(x)$ is $\mathcal{O}(g(x))$

$\Leftrightarrow$

There exist $c, x_0 \in \mathbb{R}^+$ such that $f(x) \leq c \cdot g(x)$ for all $x > x_0$

This definition helps us "categorize" the running time and the memory used by solutions using their *growth*. For example: if program read some amount N of 32-bit integers in an array, it would likely use somewhere around $32 \cdot N$ bits of memory, which is $\mathcal{O}(N)$ with some multiplicative constant $c > 32$.

If the same program also sorted those numbers using a simple quadratic procedure (e.g. repeatedly finding the "next maximum" and moving it to the end of the array) then the total number of seconds would be something like:

$$\alpha \cdot (N + N - 1 + N - 2 + \cdots + 1) = \alpha \cdot \frac{N \cdot (N + 1)}{2}$$

Where α indicates the number of seconds required to do a compare operation and an assignment, in memory. This value depends on the machine used, but we can expect it to be less than 0.0000001 in most cases. The sum is $\mathcal{O}(N^2)$, provided that we choose some multiplicative constant that is large enough (even $c = 1$ would abundantly suffice).

This hypothetical program can thus be categorized as such:

- Time Complexity: $\mathcal{O}(N^2)$
- Space Complexity: $\mathcal{O}(N)$

In APS, the **optimal** complexity is the *lowest possible complexity* of any known program which correctly solves the same problem. In the case of our hypothetical program, we can say that the space complexity is optimal since any program will use at least $\mathcal{O}(N)$ bits to store and report N integers. However, the time complexity is not optimal since there are known programs which will sort N integers in $\mathcal{O}(N \cdot \log N)$ seconds.

1.1.4 APS and Programming Languages

It's definitely possible to set up an APS competition where the participants are required to write a detailed explanation **in English** (or some other natural language) of the procedure they would use in order to solve the tasks. Doing this, though, would require a human judge who would be in charge of reading each solution, understanding what it does, deciding whether the procedure is correct, calculating its time and space complexity, and finally grading it with some score or with a rejection. This might sound like a lot of work, but it was actually (part of) how the first editions of the IOI worked.

Writing **computer programs** to solve these kinds tasks offers several advantages: there is no *ambiguity* in a formal language like C++, as opposed to a natural language like English; moreover, being able to compile and execute a program makes it much easier to *test* its correctness.

In the first editions of the IOI, the athletes were required to produce both a solution expressed in natural language and one expressed in the form of a computer program, saved on a floppy disk, so that the human grader would test the program's correctness by manually feeding input data on a keyboard and checking that the output was correct [6].

Nowadays, with the increased automation of the grading process, it's virtually always required from APS athletes to only write a program in a programming language chosen between a fixed set of allowed languages. Some automated system will take care of grading the program.

The set of programming languages which can be used in some APS contest is usually chosen by the whoever sets up the contest, but it almost always includes compiled languages such as **C**, **C++**, and **Java**. Less frequently we see interpreted languages such as **Python** and **JavaScript**, despite being some of the most popular ones in the software industry [7].

One of the reasons Python and JavaScript sometimes aren't allowed in APS competitions is that it's harder to automatically judge the asymptotic computational complexity of programs written in those languages. In fact, interpreted languages introduce an overhead [8] both in the execution time and in the required memory: this overhead is not predictable and can sometimes be so great that an optimal solution (i.e. a solution which would be in the optimal complexity class) might be mistaken for a suboptimal solution when using an automated grading system.

This problem can be addressed by increasing the size of the input data (which is akin to choosing a larger x_0 when evaluating the complexity class) and increasing the time/memory limit of the task. In this way, the constant factor introduced by the overhead of an interpreted language will become less significant.

Even though in theory this would be possible, in practice we find that low-level compiled languages (such as C, C++, Pascal) are still greatly favored in this kind of competitions. With these languages the overhead is reduced, which is akin to choosing a lower multiplicative constant c when evaluating the complexity class, and this makes it easier to "automatically evaluate" the complexity by testing the execution time and memory usage.

It's worth noting that even with low-level languages it can sometimes be hard to distinguish between, say, an $O(N \cdot \log N)$ and a $O(N)$ solution. That is, there might be an exceptionally fast "suboptimal" solution whose running time closely resembles that of an normal "optimal" solution. In general it can be very hard to grade submissions correctly [9].

Oftentimes, this happens because the input size is significant. In some cases, even just reading the input data and storing it into memory can take up a nontrivial chunk of the execution time. When this happens, the task author will usually adopt what is known as a "manager", or "grader".

By **manager** in an APS task we refer to a program that is *compiled together* with the program submitted by the athlete. This allows the task author to force certain behaviors of the submitted programs to be uniform (i.e. every athletes will do "part of" the task in the same exact way). The way this is implemented is that, when there is a manager, the problem does not tell the participants to read data, but rather to implement some specific function.

1.2 The Contest Platform

We already talked about the "contest platform" without spending too much time defining it, but this component truly represents the single most important part of any APS competition. Even though in the past there have been competitions with human graders, doing that every time requires a great effort and doesn't scale with the number of participants.

We define a **contest platform** as a software that is exposed to the contest participants via some interface: usually, the interface is web-based, less commonly it's terminal-based. A contest platform can have a range of duties, but the most important one is to **accept** the programs submitted by the participants, as long as the submission is made before the end of the contest.

Additional duties of a contest platform can include:

- **Storing** the program submitted by the participants (e.g. to make it possible for the contest managers to inspect) in some folder in the file system, or in a database
- **Compiling** the program into an executable, if it's not one already

- **Running** the program (one or more times, with some input data) if necessary
- **Grading** the program, that is, giving a grade or score to it (e.g. by using a fixed set of known testcases to verify that the program behaves like it should)
- Showing some kind of **feedback** (e.g. the score) after the participant has made a submission

Moreover, a contest platform can provide support for features that may not be meant to be used by participants. For example, providing a **ranking** that may be open only to spectators (i.e. people who are not taking part in the contest).

1.2.1 Notable Contest Platforms

As previously mentioned, in the early editions of IOI there used to be a human grader who would manually input data into the program of the contestant and judge whether the program was correct or not. This changed in the 1994 edition of IOI in Haninge, Sweden. In occasion of that IOI, an early version of an automated grading system was introduced [10].

Since 1994 a lot has changed, numerous contest platforms were created, with some notable ones becoming prominent in the APS community:

- **DOMjudge**: written in PHP, this software has been in use since 2004 in some regional contests, and since 2012 it has been used in the ICPC World Finals [11].
- **Contest Management System**: usually shortened to CMS, this software has been explicitly built to be used in the 2012 International Olympiad in Informatics [12], and it has been used in virtually every IOI since then [13]. It's written in Python.

1.3 Timeline of APS competitions

One of the earliest APS competitions can be traced back to the year 1970, when the "First Annual Texas Collegiate Programming Championship" was held at Texas A&M University. There were 8 or 9 teams participating, and 3 tasks to be solved in the Fortran programming language. From 1977 on, the contest evolved in what is known today as ICPC, a multi-tiered competition with "regionals" leading up to a world final.

In May 1989, the UNESCO initiated and sponsored the first International Olympiad in Informatics. That IOI saw the participation of 13 countries. Since then, numbers have grown considerably: the IOI 2019 counted 87 participating countries.

Among other more recent APS competitions that have risen to notability we find the Google Code Jam (GCJ) which started in 2003 and the Facebook Hacker Cup (FHC) which started in 2011. Both the GCJ and FHC are multi-tiered: they consist of several online elimination rounds and one final (offline) round. At the end of each online round the highest-ranking users are selected for the next round, until some number of finalists is chosen and invited to a final round that usually takes place in the company's offices.

1.4 History of the OII

Some countries participated in the IOI since the very first editions. One notable example is China, which actually had its own national competition already in place in 1984, five years before the IOI even started: it's not too surprising then that China is currently the country with the highest number of medals ever won in this competition.

In the case of **Italy**, clearly things went differently. Even though there are

records of sporadic participations of Italy in some pre-2000 IOIs, Italy only started officially participating in the IOI in the year 2000. This participation was sponsored by the government initially only as an experiment; this meant that 2001 was the year when for the first time Italy set up a proper funnel to select the 4 students that would go to IOI. That, was the year we finally opened the doors to every Italian high-school student who wished to compete and have a chance at representing the country at the IOI [14].

From 2000 to 2019 we can see that Italy has seen a fruitful participation in this competition. In 20 editions we were awarded 2 gold medals, 20 silver medals and more than 30 bronze medals.

In order to choose 4 students that would represent the country at IOI, the Italian government sponsored a national competition called *Olimpiadi Italiane di Informatica* (**OII**), a multi-tiered contest divided in three stages.

The first stage is a **school-level selection**, where students participate from their own high-school: at this stage, the programming problems are very simple and the students are usually not required to write any code, they rather need to be able to just read and comprehend a simple program in order to predict its behavior.

In its 2018/19 edition, the school-level selection was attended by a total of **13625** athletes.

The second stage, reserved to the highest ranked students from the school-level selection, is a **district-level** selection, where *for the first time* students are asked to write and submit programs, in a more "classic" APS setting. In this early stage the students are not required to find optimal solutions, even just working solutions are usually accepted.

Every year, a set of "districts" is chosen by the OII committee: at first, the obvious way of splitting up the nation in districts was to simply consider Italy's 20 geographical regions; this worked well in the early editions but, as the competition gained traction, it was changed to reflect the fact that

some regions consistently had a higher participation rate than others, or they simply had a larger population.

In its 2018/19 edition, the district-level selection was attended by a total of **1358** athletes. About 1 in 10 compared to the school-level selection. The number of students invited to this stage was actually 1798, but since the events are taking place months apart from each other, and since many students decide to not attend, a significant percentage of students (24.5% in this case) just doesn't show up to the contest.

After this second stage is done, the best students are invited to a **national final contest** that closely resembles the IOI, but at a smaller scale. As opposed to the district-level contest, in this third stage of the competition, the students are required to find the optimal solution for the tasks proposed. If a solution is correct it will get some points, but not the full score (100 points) unless it is also optimal.

In its 2018/19 edition, the national final contest was attended by **95** athletes.

1.4.1 Going "teams": the OIS

The *Olimpiadi di Informatica a Squadre* (**OIS**) aims to be a team-oriented version of the already-existing Italian Olympiads in Informatics. A relatively new project, the OIS introduces many differences with its "individual" counterpart, one of the most important ones is the almost complete decentralization of the contests: there are four **completely online** rounds that lead to an onsite final.

The initiative started in February of 2010 in Italy as an idea by professor Giulio Angiani: only 7 teams were participating but, as the initiative started to catch on, more schools joined the project [15].

In the 2019/20 edition, **506** teams from 132 schools participated in the OIS.

2 OII: Software and Procedures

In the OII (Olimpiadi Italiane di Informatica) and in its team-based counterpart OIS (Olimpiadi di Informatica a Squadre) we use a number of different procedures, as well as software systems, to perform various functions throughout the phases of the competition.

2.1 Human Resources

In each participating school we identify a **RS**: "referente scolastico", or school representative. This person is usually a teacher in that school who will be in touch with the support staff before the start of the contest to handle the logistics and the distribution of the problem set to the students, and who will be available for the whole duration of the contest to relay possible clarifications (staff → students) or questions (students → staff).

At the next level, that is, for each district, we similarly identify a **RT**: "referente territoriale" (district representative). An RT is usually also a RS, but on top of the school-selection held in November, he/she is additionally in charge of the district-level selection which takes place in April.

During each contest, there are often questions coming from the students or even the representatives themselves. For this reason, in each contest there is always a group of **support staff** which is available for the entire duration of the competition to answer questions.

2.1.1 Communication Channels

To relay information between the staff and the representatives, we mostly make use of existing software:

- Chat Groups (*Telegram*, *WhatsApp*)
- Hosted Forum software (*JForum*)

Before and during the school-level contest, we keep a forum where RSs can open new topics or reply to existing ones.

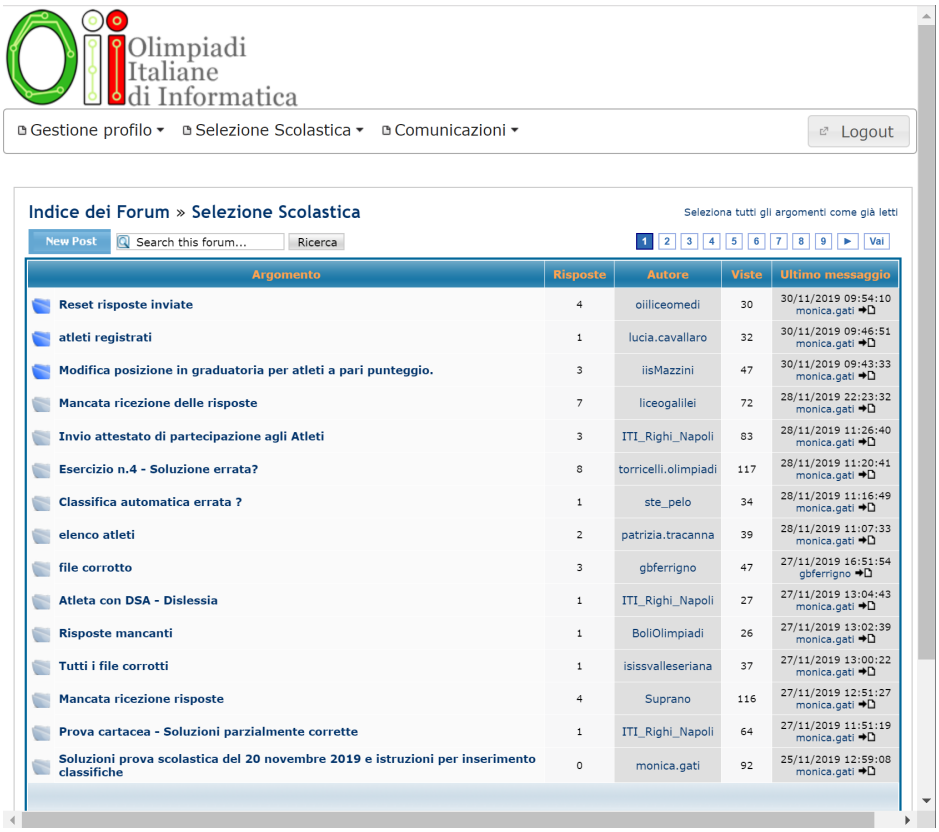


Figure 2.1: The support forum for the school-level contest

Before and during the district-level contest we keep a chat group with all the RTs so that the technical staff can help them with logistics.

The main advantage of a forum over a group chat is that the information is kept organized in threads which are less likely to be overlooked by others who might be facing the same problems. For example, a thread could host a discussion on the interpretation of a specific problem from the contest, or about a common issue that many schools are facing, and so on. This helps with reducing the amount of "same-questions".

2.2 CMS: software architecture

The software that we use the most is undoubtedly CMS, the contest platform that we already mentioned in 1.2.1 and that we use both in the national finals of the OII and in all the rounds of the OIS.

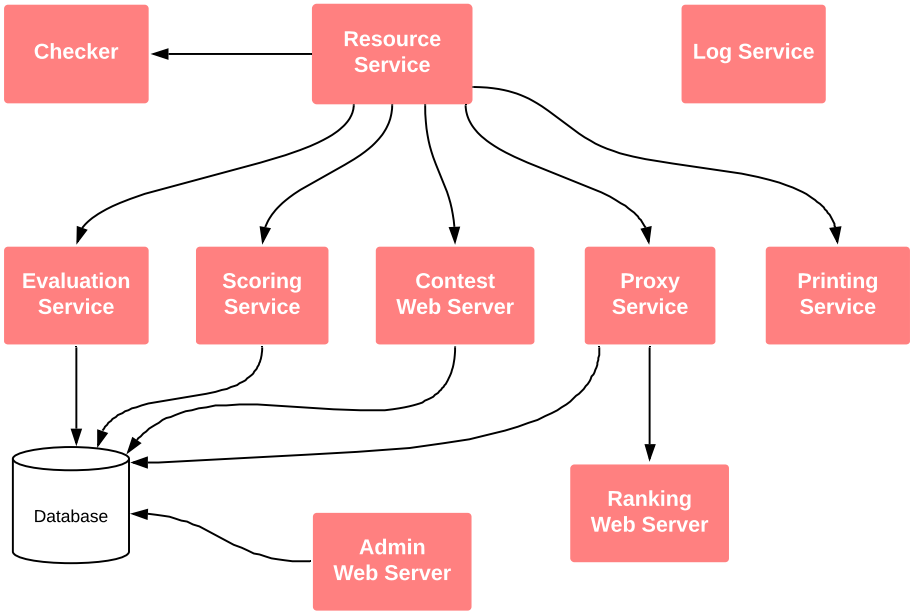


Figure 2.2: A diagram of the CMS components

This software is very versatile and, as we often find ourself doing, it can be extended to serve new features and purposes. It's written in the Python

programming language and it's split in different executables that can be easily distributed in a network and scaled up: for example, the throughput of graded submissions can be improved by spawning more Worker processes, which will handle more jobs at the same time.

We will now give a brief overview of the most important components of CMS.

2.2.1 The `EvaluationService` process

This component is tasked with orchestrating the evaluation of submissions as they trickle in the system. Whenever a new submission is recorded in the database, the EvaluationService process fires up and creates "batches" of jobs which will later be performed by Worker processes.

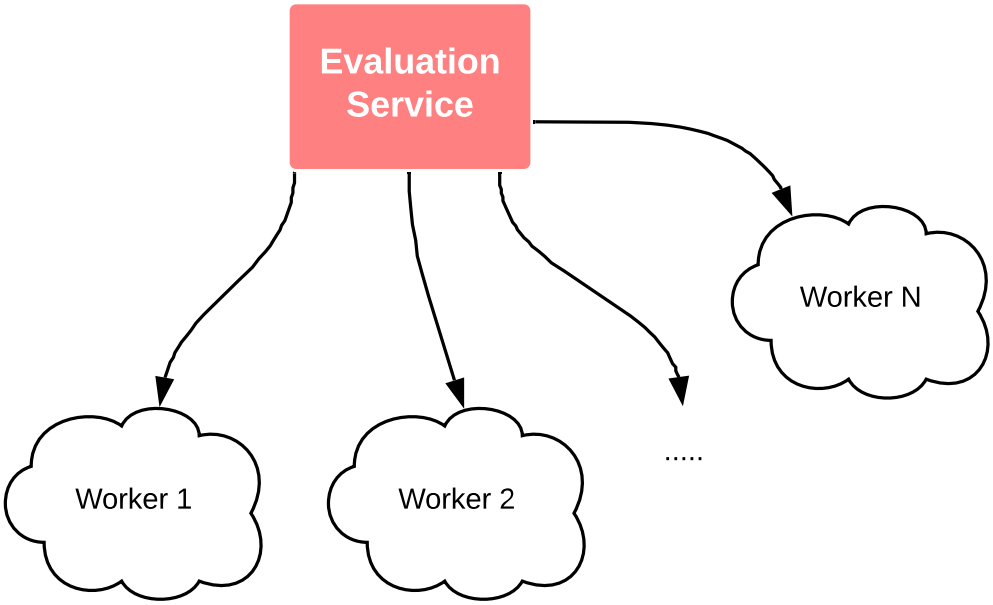


Figure 2.3: The EvaluationService process orchestrates jobs for Workers

2.2.2 The `Worker` process

The Worker component is the one in charge of running the actual grading process of a submission, that is: running the correct compilation commands depending on the programming language of the submission, executing the untrusted program in a sandbox (enforcing time and memory restrictions) and checking that the output produced is valid.

Worker processes are stateless, they receive everything they need (source code, input data, and so on) from the EvaluationService process over the network. In order to perform more efficiently, each worker keeps a cache folder where files are indexed by their hash.

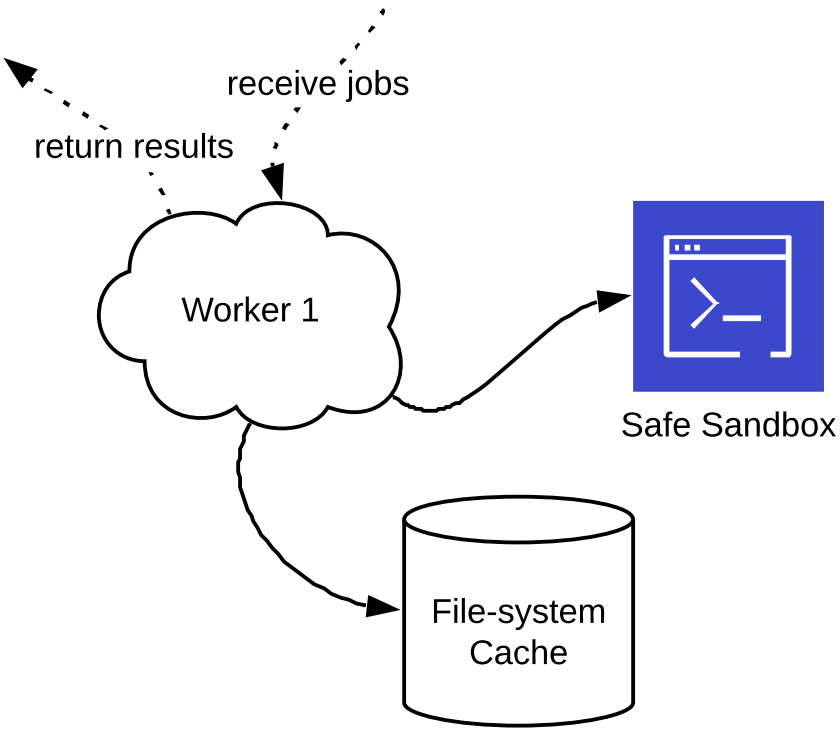


Figure 2.4: The Worker grades a submission, using a cache to reduce requests to EvaluationService over the network

2.2.3 The `ContestWebServer` process

This component implements a web server, so it listens for HTTP connections by clients (mostly the contestants' browsers) and responds accordingly. The role of this process is to accept submissions and provide feedback to the contestants.

Contestants access this component to read task statements, to submit solution to tasks (and receive feedback) and to submit questions to admins (and receive answers).

Similarly to Workers, this component can also be spawned multiple times in order to serve a higher amount of contestants.

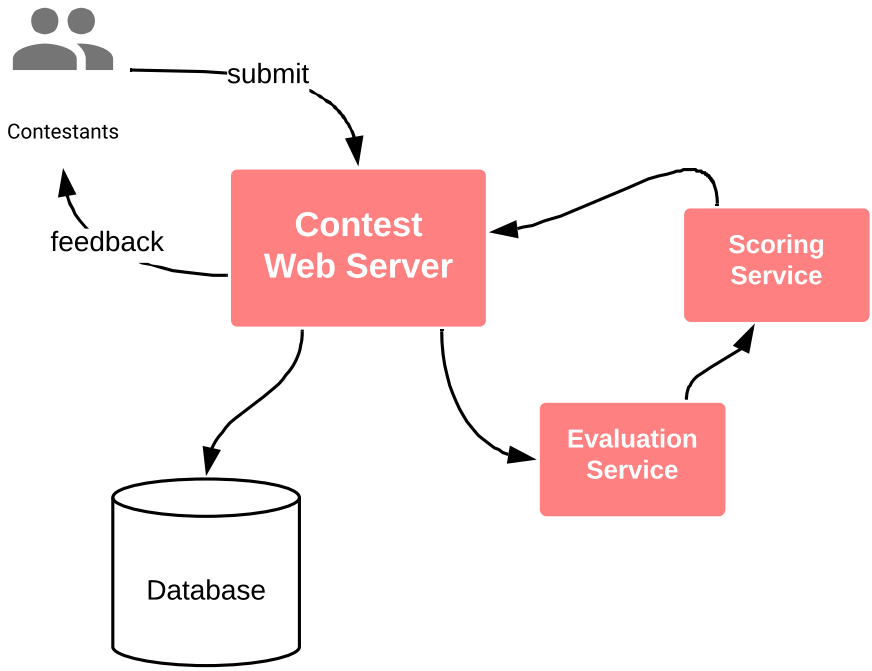


Figure 2.5: The ContestWebServer process accepts submissions and shows feedback to the contestants

2.3 CMS: user experience

In CMS there are three "actors": contestants, admins, spectators. These interact with the system via different interfaces such as the browser and the command line.

What follows is an overall schema of the interaction between the actors and the various components of the CMS system.

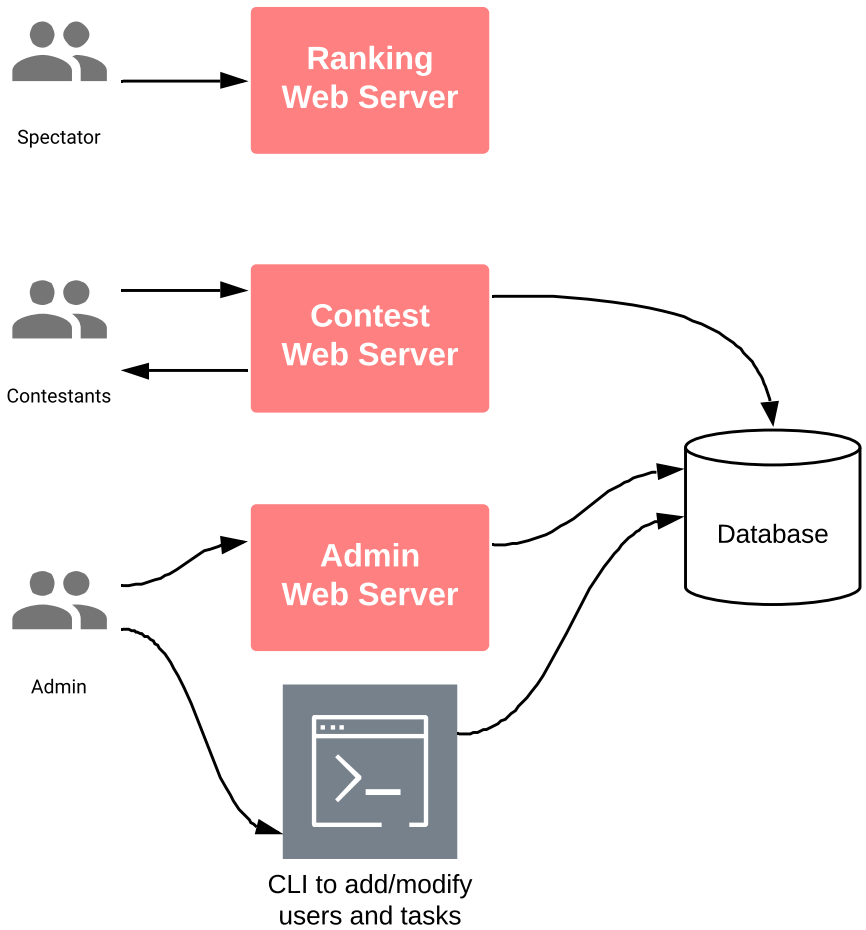


Figure 2.6: Actors interact with the CMS system

2.3.1 Contestant: submitting a program

The task statement page immediately shows to contestants the information that they will most likely need: time limits, memory limits, the full compilation commands. To access the actual statement, an additional step is necessary: clicking the "Download statement" button.

Instead of embedding the statement in the page itself, CMS opts for requiring a separate step. Downloading the statement means permanently storing it on the device, which can be a wise thing to do in case of online contests since the Internet connection could malfunction or the server could become overloaded at some point during the competition.

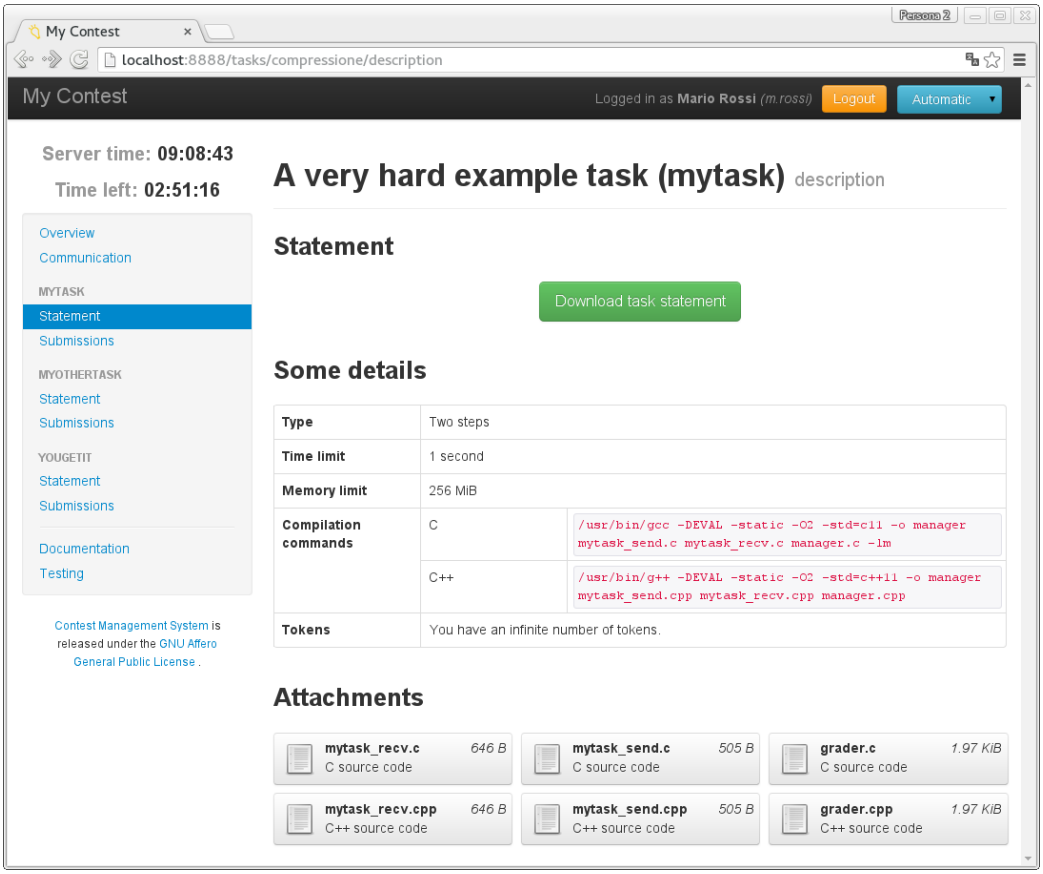


Figure 2.7: Screenshot of the CMS task statement page

Since CMS was explicitly designed for a competition of International level, when the average user (contestant) uses CMS, he or she is likely to have experienced participating in a coding competition in the past, in their own country.

Moreover, during the IOI there is always a "practice session" where the contestants can get accustomed to the contest platform, before the start of the actual competition.

As for what concerns **submitting** a program for evaluation, the submission page of CMS always shows the list of the submissions previously made by the contestant. It also shows multiple options to submit (single files or zip).

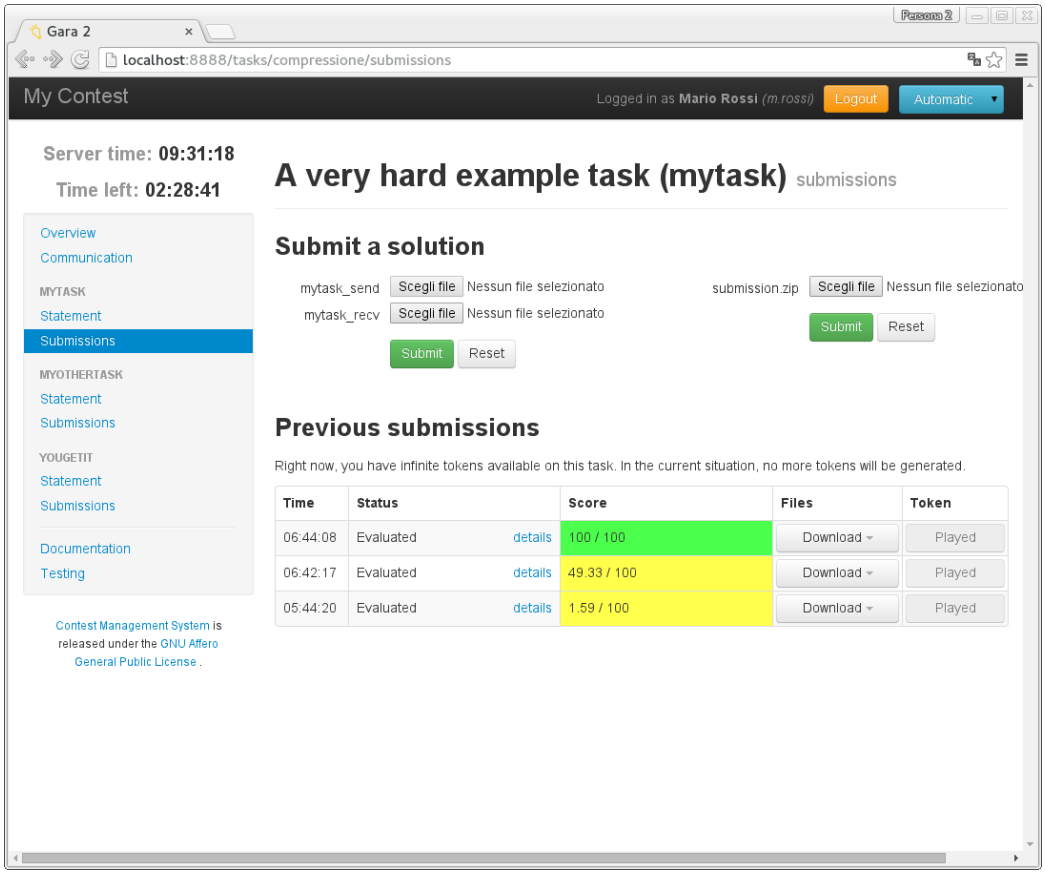


Figure 2.8: Screenshot of the CMS task submission page

2.3.2 Admin: setting a task

Normally, the admin prepares a file-system representation of the task, which will be then *imported* in the database using the CMS command-line interface.

```
christmas/
├── att
│   ├── christmas.c
│   ├── christmas.cpp -> ../att/christmas.c
│   ├── christmas.pas
│   ├── input0.txt -> ../testo/christmas.input0.txt
│   ├── input1.txt -> ../testo/christmas.input1.txt
│   ├── output0.txt -> ../testo/christmas.output0.txt
│   └── output1.txt -> ../testo/christmas.output1.txt
├── gen
│   ├── GEN
│   ├── cattivo.txt
│   ├── generatore.py
│   ├── limiti.py
│   └── valida.py
├── sol
│   ├── exponential.cpp
│   ├── greedy_edomora97.cpp
│   ├── soluzione.cpp
│   └── unodue.cpp
├── task.yaml
└── testo
    ├── christmas.input0.txt
    ├── christmas.input1.txt
    ├── christmas.input2.txt
    ├── christmas.jpg
    ├── christmas.output0.txt
    ├── christmas.output1.txt
    ├── christmas.output2.txt
    ├── english.pdf
    ├── english.tex
    ├── logo.pdf -> ../../../util/static/logo/asy/logo.pdf
    └── testo.pdf -> english.pdf

4 directories, 28 files
```

Figure 2.9: The CMS file system representation of a task

After importing tasks in the database the admin can use the `AdminWebServer` component to perform actions such as managing the task evaluation, enabling or disabling `Workers` at run time, and so on.

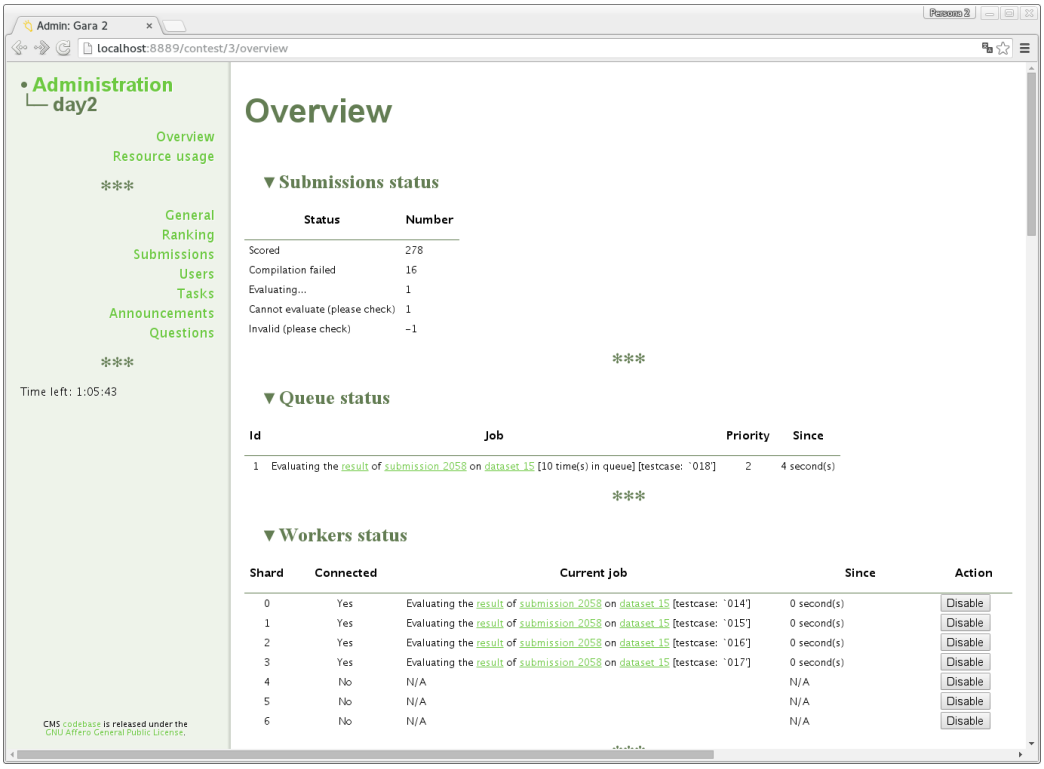


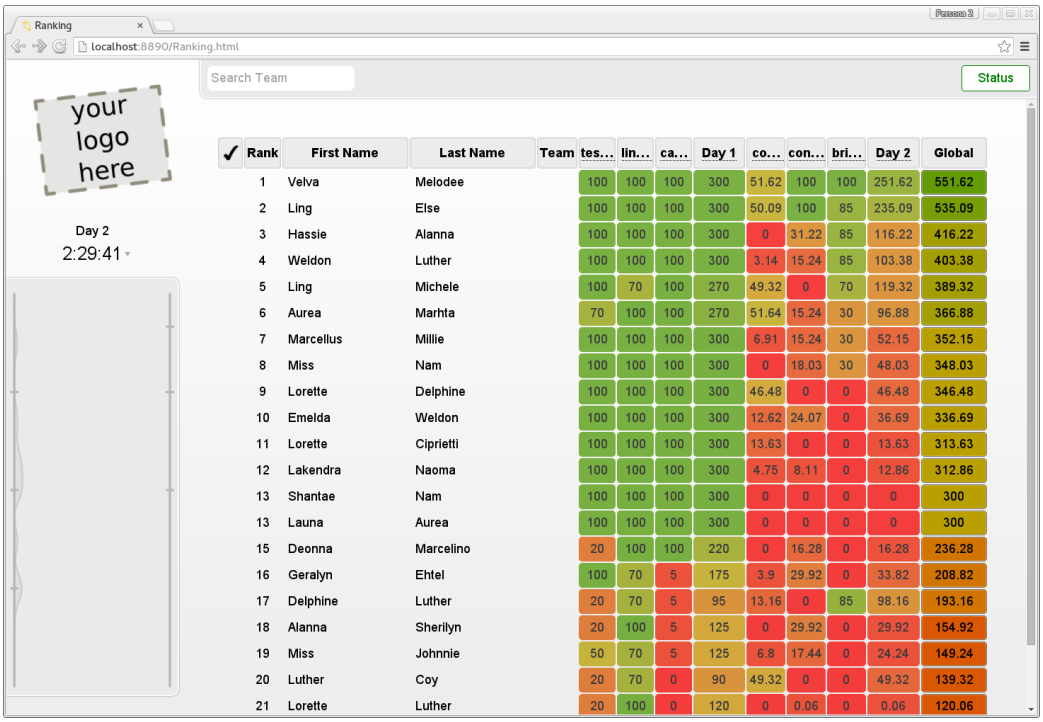
Figure 2.10: Screenshot of the CMS administration page

2.3.3 Spectator: using the ranking

When a spectator wishes to get information on a running contest, he or she can access the `RankingWebServer` component, which is usually exposed on a public URL that anyone with an Internet connection can reach.

This component has a web interface that allows getting specific information on the participating users, sorting them, searching for contestants belonging to a specific country or team, "following" some contestants so that they are highlighted in the ranking, and accessing the history of submissions for a

specific contestant.



✓	Rank	First Name	Last Name	Team	tes...	lin...	ca...	Day 1	co...	con...	bri...	Day 2	Global
	1	Velva	Melodee		100	100	100	300	51.62	100	100	251.62	551.62
	2	Ling	Else		100	100	100	300	50.09	100	85	235.09	535.09
	3	Hassie	Alanna		100	100	100	300	0	31.22	85	116.22	416.22
	4	Weldon	Luther		100	100	100	300	3.14	15.24	85	103.38	403.38
	5	Ling	Michele		100	70	100	270	49.32	0	70	119.32	389.32
	6	Aurea	Marhta		70	100	100	270	51.64	15.24	30	96.88	366.88
	7	Marcellus	Millie		100	100	100	300	6.91	15.24	30	52.15	352.15
	8	Miss	Nam		100	100	100	300	0	18.03	30	48.03	348.03
	9	Lorette	Delphine		100	100	100	300	46.48	0	0	46.48	346.48
	10	Emelda	Weldon		100	100	100	300	12.62	24.07	0	36.69	336.69
	11	Lorette	Ciprietti		100	100	100	300	13.63	0	0	13.63	313.63
	12	Lakendra	Naoma		100	100	100	300	4.75	8.11	0	12.86	312.86
	13	Shantae	Nam		100	100	100	300	0	0	0	0	300
	13	Launa	Aurea		100	100	100	300	0	0	0	0	300
	15	Deonna	Marcelino		20	100	100	220	0	16.28	0	16.28	236.28
	16	Geralyn	Ehtel		100	70	5	175	3.9	29.92	0	33.82	208.82
	17	Delphine	Luther		20	70	5	95	13.16	0	85	98.16	193.16
	18	Alanna	Sherilyn		20	100	5	125	0	29.92	0	29.92	154.92
	19	Miss	Johnnie		50	70	5	125	6.8	17.44	0	24.24	149.24
	20	Luther	Coy		20	70	0	90	49.32	0	0	49.32	139.32
	21	Lorette	Luther		20	100	0	120	0	0.06	0	0.06	120.06

Figure 2.11: Screenshot of the CMS ranking page

2.4 Using CMS in the Italian Olympiads

We use CMS, sometimes with slight modifications, both at the *national finals* of the OII and in the *online rounds* of the OIS.

2.4.1 CMS in the national finals of the OII

In the national finals of the OII, as explained in 1.4, we normally have around 100 contestants and the contest is offline, so we have complete control over the software environment that the users will be running during the contest.

In this setting, we are able to circumvent irregular behavior by adopting a

set of measures such as:

- Disabling USB ports of the contest workstations.
- Using a firewall to block communication outside the contest arena and within the arena (i.e. the different workstations mustn't be able to ping each other).
- Use information on the contestants (e.g. the school where they come from) to distribute them as far as possible from other contestants that they might know.

The contestants get registered into the CMS database by using custom-made scripts. The OII staff keeps all the scripts in a private git repository so that they are always accessible to whoever is physically running the contest in that year.

After all, the national final "setting" is very similar to the IOI one, so CMS proves to be especially appropriate for this specific use-case.

2.4.2 CMS in the online rounds of the OIS

Online rounds present challenges that aren't normally found in offline rounds. Most evidently, there is the fact that when contestants are connected over the Internet they implicitly have access to online resources that might not be allowed in the contest (such as: pre-written code, or messaging systems that allow contestants to interact with other people during the contest).

This made it sensible to put in place some reasonable "security measures" as a way to discourage students from cheating.

Most notably, we operated some changes to the CMS software that are aimed at detecting whether the user is connected to the Internet. On top of that

we also put in place a plagiarism-detection pipeline that, after the contest has ended, analyzes the submissions to find similarities between programs.

In order to restrict access to the Internet while allowing traffic directed towards the contest platform, we prepared a document called "Local Setup Guide" aimed at RSs, in this document we explained in detail how to configure the network in such a way that the only traffic permitted is towards a single website. Enforcing this "whitelisting" proved to be extremely challenging: we haven't reached full compliance and we don't expect to reach it soon.

2.5 The district-level selection

For almost two decades the district-level selection was held by delegating most of the contest responsibilities to an external entity, a company, who took care of all the logistic aspects of the competition, like: distributing tasks, creating and distributing user credentials, collecting the submitted programs, grading the programs, delivering the feedback.

This changed with the development of Terry. For now, let's see in detail how the contest used to be handled by this external entity.

2.5.1 Task distribution

The OII staff prepared the contest tasks, each with a fixed set of test cases. This would then be sent to the company in charge of the actual task distribution procedure.

In order to distribute the tasks to the students, the delegated company prepared a virtual machine (VM) with a FTP server running in it. This virtual machine was distributed to each district as a virtual application file. The RTs would then boot up the VM and access an admin interface that acted as

a gateway between the external company and the school (i.e. the district).

This FTP server was used both for receiving the tasks, and for collecting the programs submitted by the students.

2.5.2 Collection of the programs

The collection of the programs developed by the contestants was handled by the same FTP server running on the server workstation. In order to submit a program, a student was expected to access a specific IP address from his or her browser, and then upload both the executable and the source code of their program.

The server machine collected the file in a folder that, through the FTP server, was accessible to the external company who would then (after the end of the contest) collect the files remotely.

The server machine did not provide any feedback about the program's results to the contestants. Nothing was executed: the FTP server just collected the files.

2.5.3 Delivering Feedback

Once the external company collected all the programs, well over the end of the contest, the actual grading process started. This process would sometimes take more than 2 weeks since there would be a lot of cleanup work involved with the students' submissions, and since among all those programs some are bound to be wrong in the sense that they run forever (the rules stated that a submission will be left running for up to 5 minutes for each testcase before manually halting it).

After this time passed, the feedback would be published in the form of a PDF file containing the students who passed to the national finals.

This feedback system was not comparable to CMS's, since the latter allowed contestants to immediately understand when their program was not correct.

It's important to see that, without a proper feedback system, contestants are left in the dark of any potential mistake in the input-output logic of their programs. This means that any small typo (e.g. in the file name) or insignificant mistake in reading or writing data (think about uppercase vs lowercase) would greatly penalize a contestant that would otherwise have a solution with the correct logic.

With a feedback system like CMS's, the contestant has a good chance at removing any of those bugs that aren't related to the algorithms and data structures of their programs.

2.6 Using CMS in the district-level selection?

The question that comes naturally is: why not use CMS in the district-level selection? After all, we saw that it's possible to slightly tweak this software to serve new purposes, as we did in the OIS.

2.6.1 Strict time limits

The main reason for not using CMS in the district-level selection is its nature of having strict time limits for the evaluation of each submission.

In fact, since in the district-level selection the participants are often facing their first ever real APS competition, the tasks **do not** require an asymptotically optimal solution to be solved: they rather require any correct solution that is reasonably fast (e.g. not with an exponential complexity if the best possible is linear).

Clearly, since there are only a finite number of Worker processes, it would be

detrimental to the throughput of EvaluationService to **not enforce** a strict time limit: some contestant could submit a program that *does not halt* and would permanently occupy a Worker.

Even having a "very large" time limit can be enough to completely clog the evaluation queue and disrupt the live feedback for everyone.

2.6.2 Not uniform server performance

Another reason for not using CMS in the district-level selection is that having a CMS instance in place of a simple FTP server can greatly exacerbate the differences in performance between the workstations that are available in each district.

If two districts use two server workstations that differ too much in terms of performance, we might be giving an unfair advantage to the contestants in the "faster" district.

3 Design of Terry

This chapter will be devoted to solving the problems identified in section 2.5 for what concerns the *district-level* selection of the Italian Informatics Olympiads. We saw in 2.6 why CMS was not apt to being used for this specific contest phase. We will now explicitly list all the requirements that are needed in the district-level selection and we will see how we designed a new software to overcome the limitations that we used to face in the past.

3.1 Functional Requirements

As we mentioned, the district-level selection of the OII is arguably the most delicate phase because of its decentralized nature. We will now list the specific requirements that we addressed in our new software design.

3.1.1 Suboptimal complexity is fine

Previously we said that in the district-level selection we usually **do not** require that the programs submitted are optimal. Or, at the very least, we expect that a suboptimal program should score all or nearly all points.

For this reason, as it was the case in the district-level contest until this point, we should design a system that does not enforce strict time limits.

3.1.2 Decentralized contest

The district-level selection is held on the same day in more than 50 different locations throughout Italy. In each of these districts, a whole contest is organized by the RTs, high-school professors which are tasked with preparing the workstations and other logistic things, like providing printing facilities when necessary. The decentralized nature of this contest implies an inherent difficulty in the management of it, since each location has a different situation in regards to connectivity, computing power of the server workstation.

Figure 3.1: In its 2019 edition, the OII identified 51 districts

3.1.3 Unstable or unreliable connectivity

In Italy, connectivity can be slower or more unreliable compared to other countries, this is true especially in the south. Even though the situation seems to be improving as of late, we still can't expect all districts to have a reliable Internet connection, so we should take that into account when designing the system. Requiring districts to use Internet could mean introducing some unfairness factor in the competition.

It is true that the OIS makes use of an Internet connection but, as we already pointed out, the OII holds a more important role as it is a funnel to the IOI. For this reason, we should design a system that guarantees a fair contest to the widest extent possible.

3.1.4 Easy to setup

Since this system is going to be set up by RTs, like the old one that was provided by an external company, we need to make sure that the setup process is easy enough as to be feasible by people that are likely to not be comfortable using a command-line interface.

Having it seen working well in the past, we could opt for distributing a virtual machine application and, instead of bundling just a FTP server, we could bundle *Terry* inside the VM.

3.1.5 Immediate feedback to users

The main reason we developed Terry was to improve the current situation in the district-level contest. The current situation was that students received feedback on their performance *up to two weeks after the end of the contest*. The new requirement in this direction is that the feedback should be given

immediately, similarly to what happens with CMS, that is, *up to a few seconds after the program is submitted* in the system.

3.1.6 Intuitive interface

This might seem like an obvious requirement, since no one who would design a non-intuitive interface on purpose. However in our case there is a very specific need for an interface which *guides the user into doing the right thing.*

The students who successfully pass the school-level selection and reach the district-level contest are likely to be in their first ever experience in a programming contest. We decided for this reason to actively hide all the buttons and "choices" that a user might have to make unless it made sense to show them [16]. For example, the button to access the "list of all the submissions" is not visible unless the user has made at least one submission.

This kind of discoverable interface has a good chance at reducing the confusion that new participants might face.

3.1.7 Remotely Inspectable

Since, again, this is a critical part of the selection funnel for the Italian participation in the IOI, we need to be extra safe. For this reason, another requirement is to arrange some way to *inspect* the system and its data, for any of the districts.

3.2 User Interface Requirements

In this section we will show the early mock-up graphical interfaces that we produced as part of the initial design of the system.

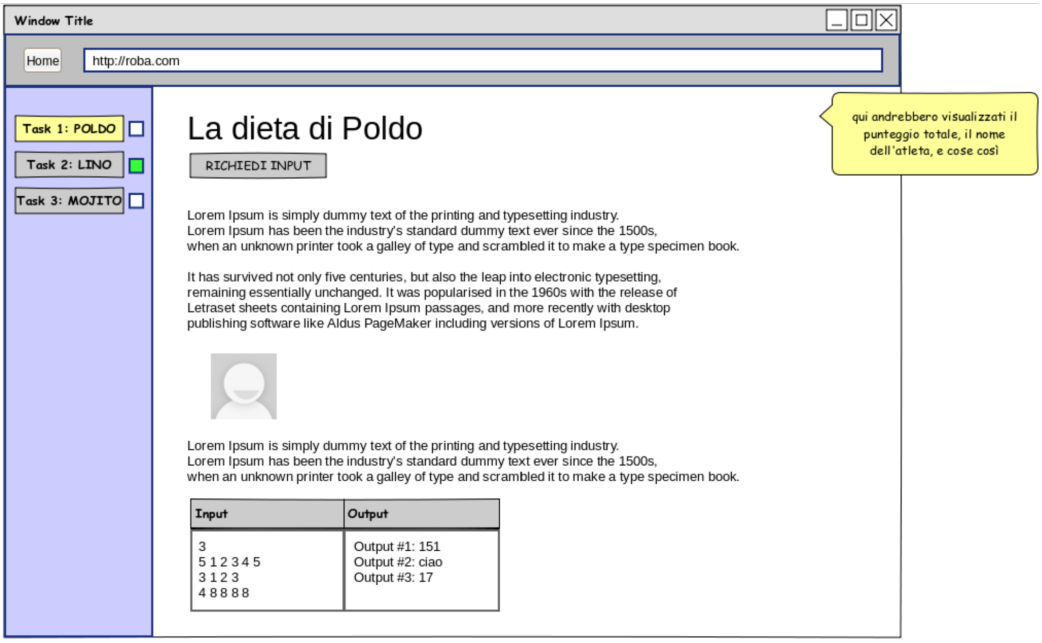


Figure 3.2: The contest page seen by the student

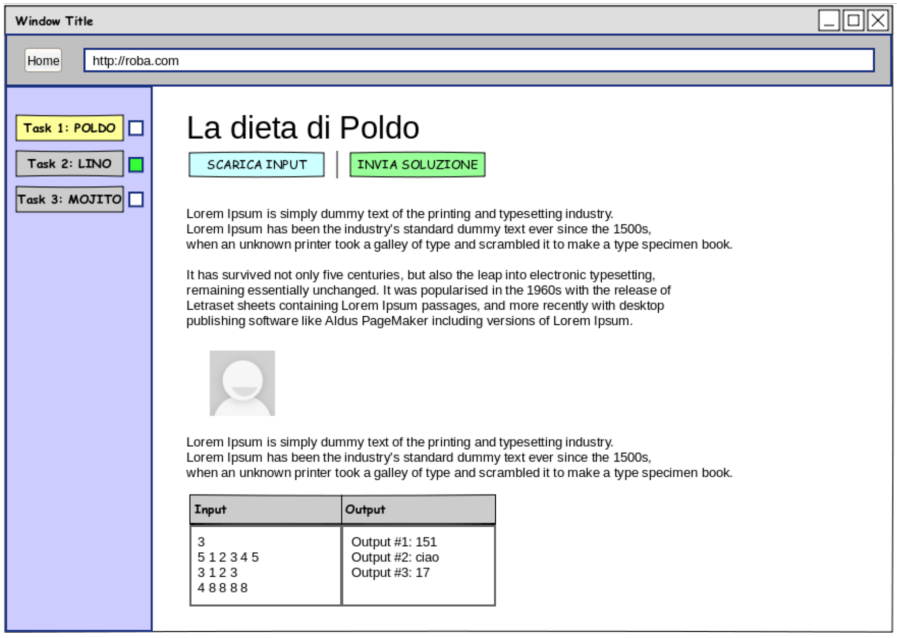


Figure 3.3: After the students clicks on "Request input" two new buttons become available: "Download input" and "Upload solution"

At this point the user can:

- Download the input file;
- Inspect it with a locally-installed text editor;
- Write a program to compute the correct output.

Computing the correct output "manually" is an option, but virtually always unfeasible due to the nature of the problems and the input size.

Once the correct output has been computed, the solution can be submitted to the system for grading. In order to do so, the user will click on the "Upload solution" button.

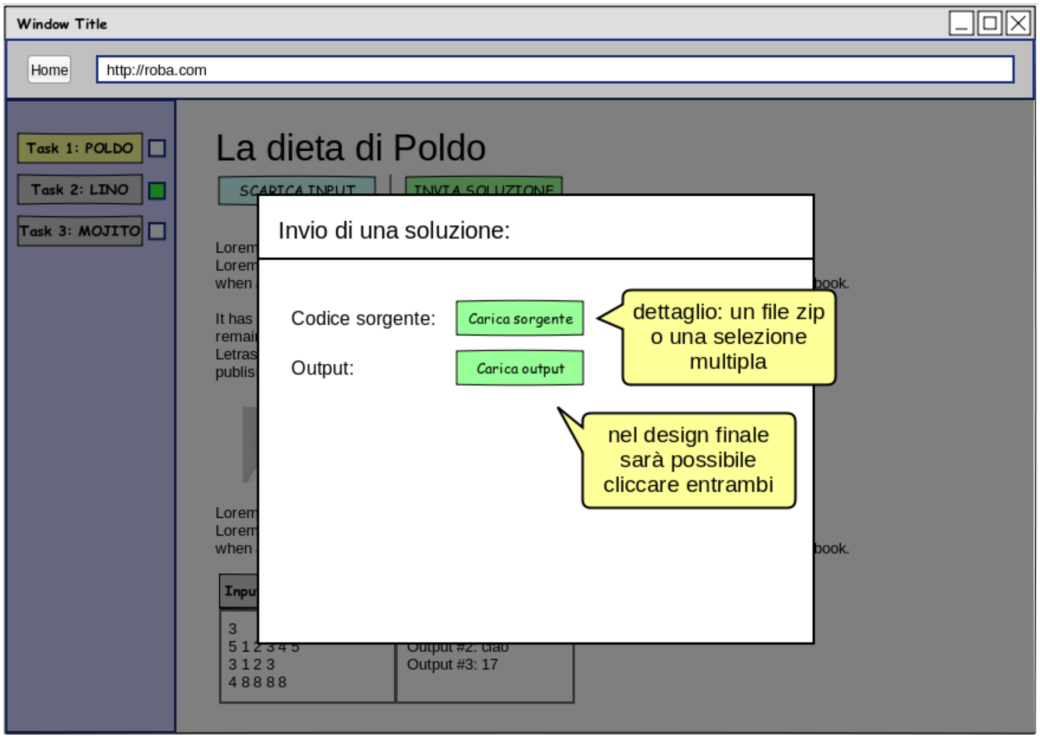


Figure 3.4: After computing the output, the students can upload the solution by clicking "Upload solution"

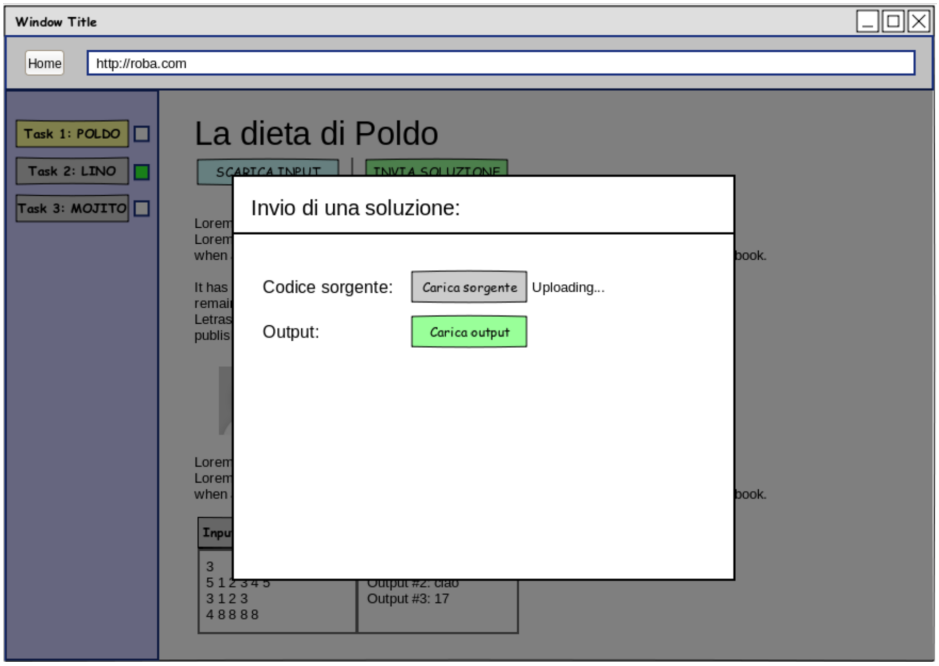


Figure 3.5: The upload of the source code is in progress

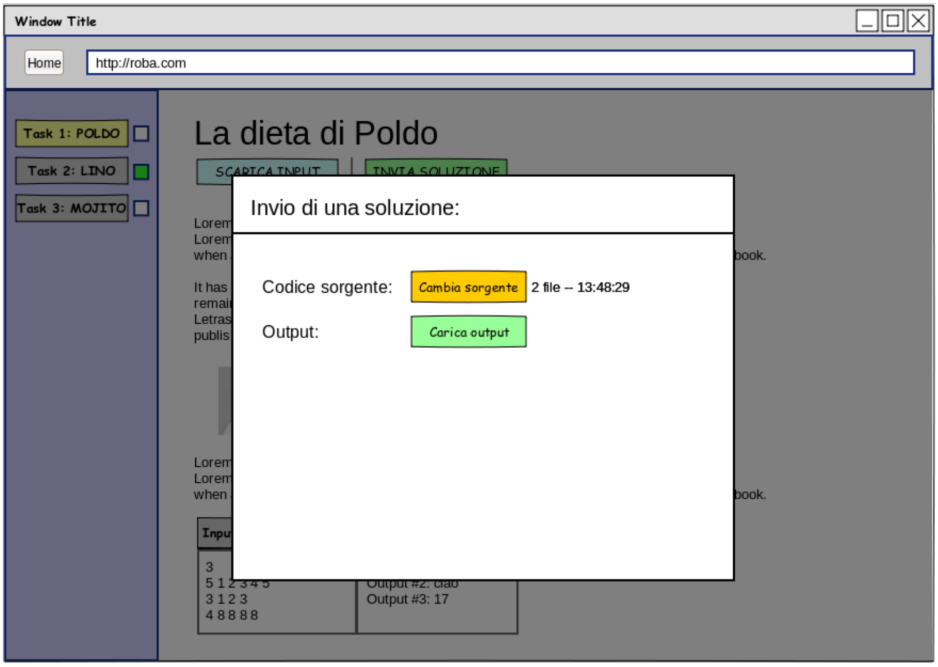


Figure 3.6: The source code is uploaded

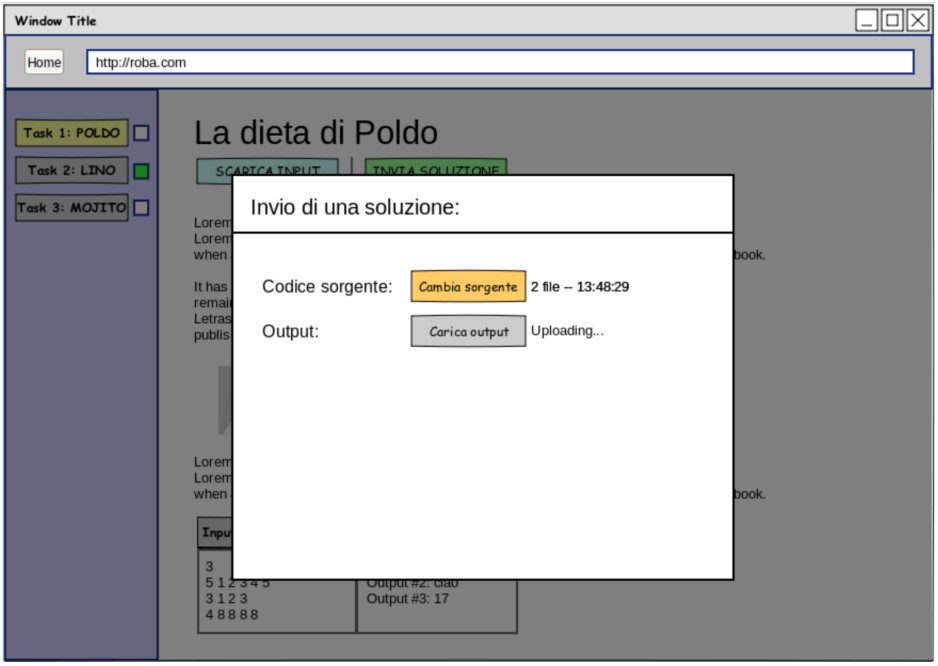


Figure 3.7: The upload of the output file is in progress

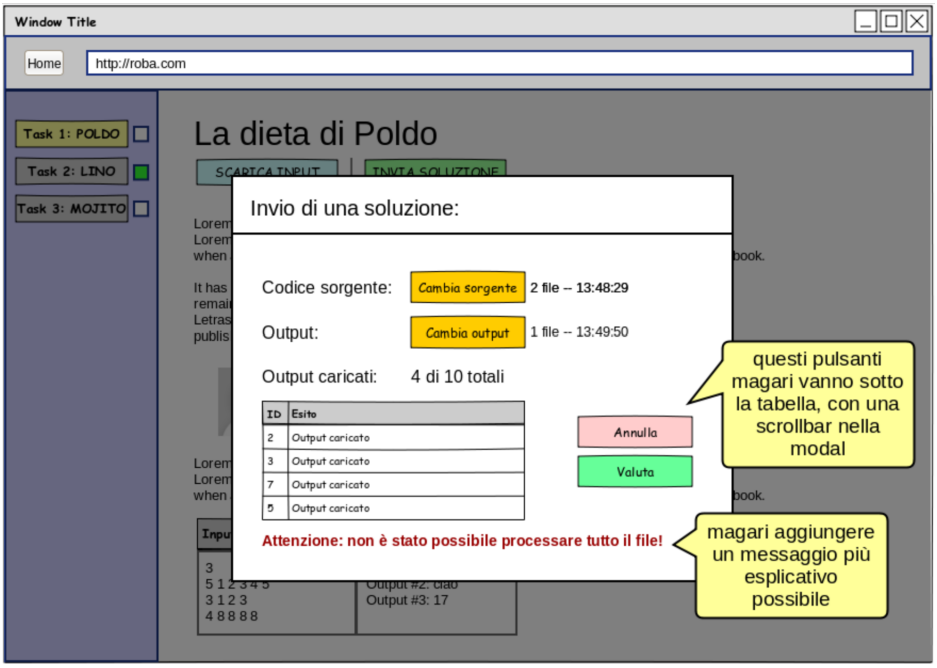


Figure 3.8: The output file is uploaded

The form lets the user upload the source code and the output file, in any order. Once *both files* are uploaded, the form will show an overview of the testcases found in the output file: at this point the user can still change their mind and re-upload one or both of the files.

If the user uploads a wrong *output file* (e.g. from a previously solved problem) the mistake will be clear from the list of testcases so the user can still correct the mistake.

If the user uploads a wrong *source code* (e.g. from a previously solved problem) the mistake will be clear from the timestamp (last modified) next to the "Change output" button. In fact, the final implementation of this user interface will show a **relative time** instead of an absolute one. This means that the user will see the last modification time as a string that looks like "*2 minutes ago*" instead of a dull-looking "13:48:29", thus making it noticeable when the file is not the right one.

Once the user is satisfied with both files submitted, they can finalize the submission by pressing the "Evaluate" button.

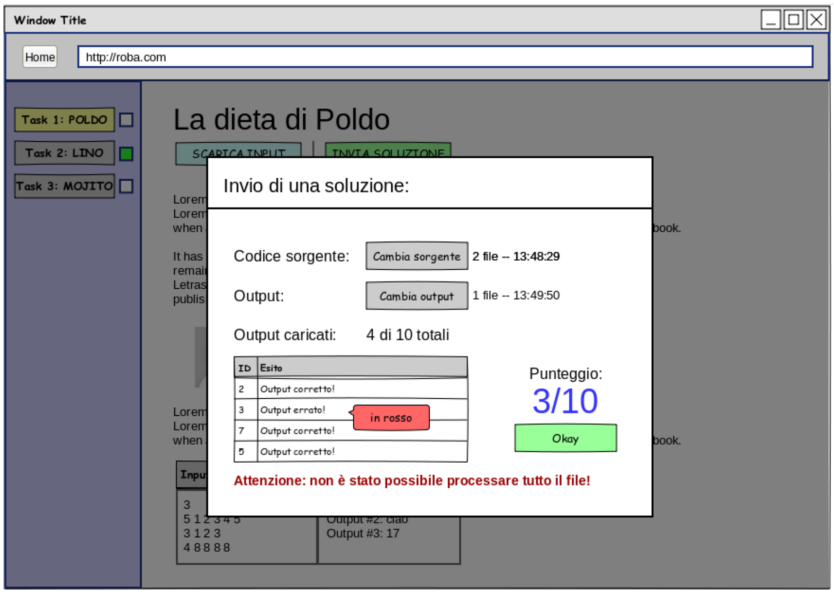


Figure 3.9: The evaluation result is presented to the user

An important requirement is: the input file must **not** be reusable. After the feedback is given, the user will know for each testcase whether it was correct or not; this means they could manually edit the output file to match the feedback (think of problems where the output is either "`YES`" or "`NO`").

For this reason, if the user finalizes the submission, then they will have to request a new input in order to submit again.

3.3 Use Cases

We will now analyze the specific use cases by looking individually at the various actors of the system.

Figure 3.10: Use case diagram of the system

3.3.1 Contestant

The contestant requests an input and uploads a solution. The following state chart represents the possible actions that the contestant can perform:

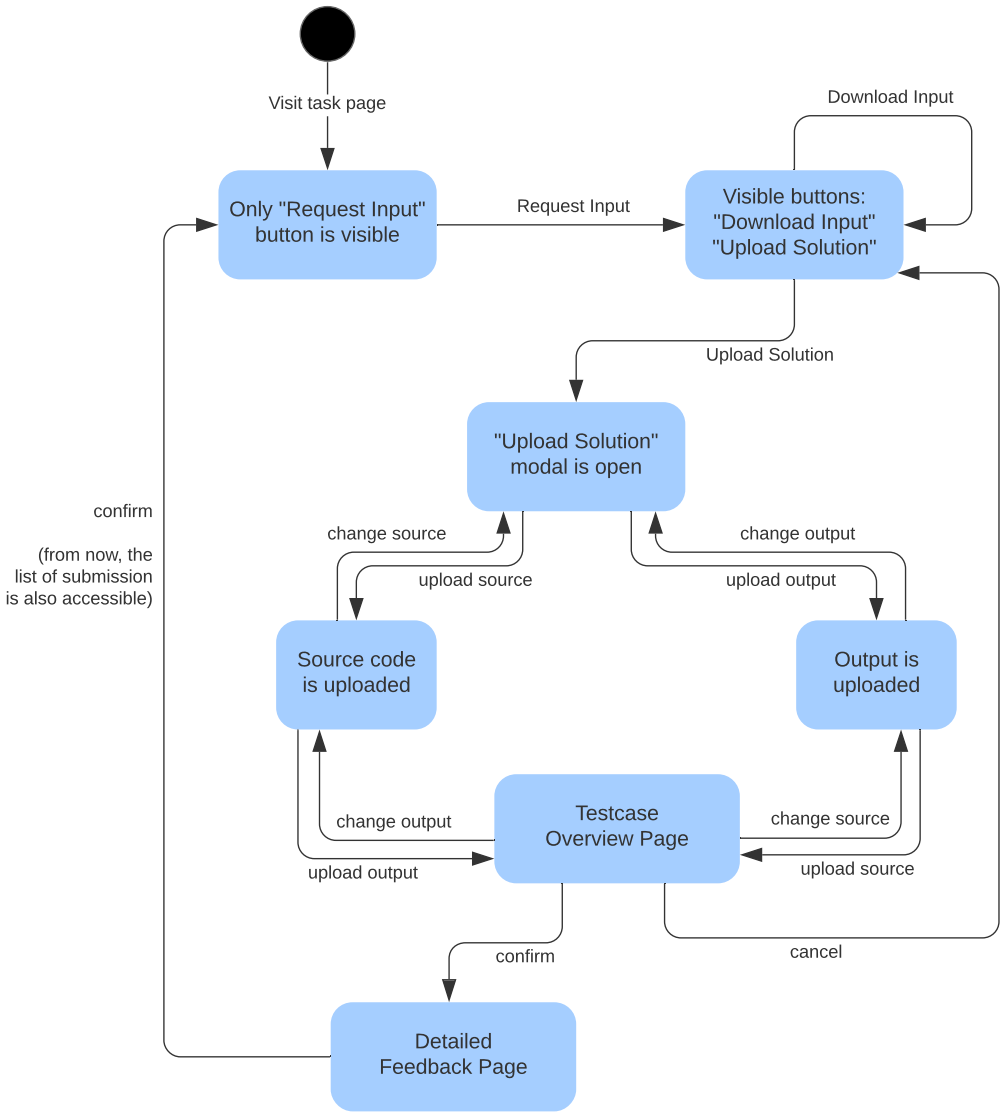


Figure 3.11: State diagram showing the “request input” flow

3.3.2 RT

Each RT will take care of setting up Terry in his or her district. Specifically, an RT will access a reserved area of the OII website and download the virtual application. Once the RT downloaded the file, they will boot up the VM on some workstation that is visible from all the "contestant" workstations.

After the contest is finished, the RT will access a special page in Terry that lets them download a zip file with the VM data. This zip file will then be uploaded by the RT in the same reserved area of the OII website where the VM was first downloaded.

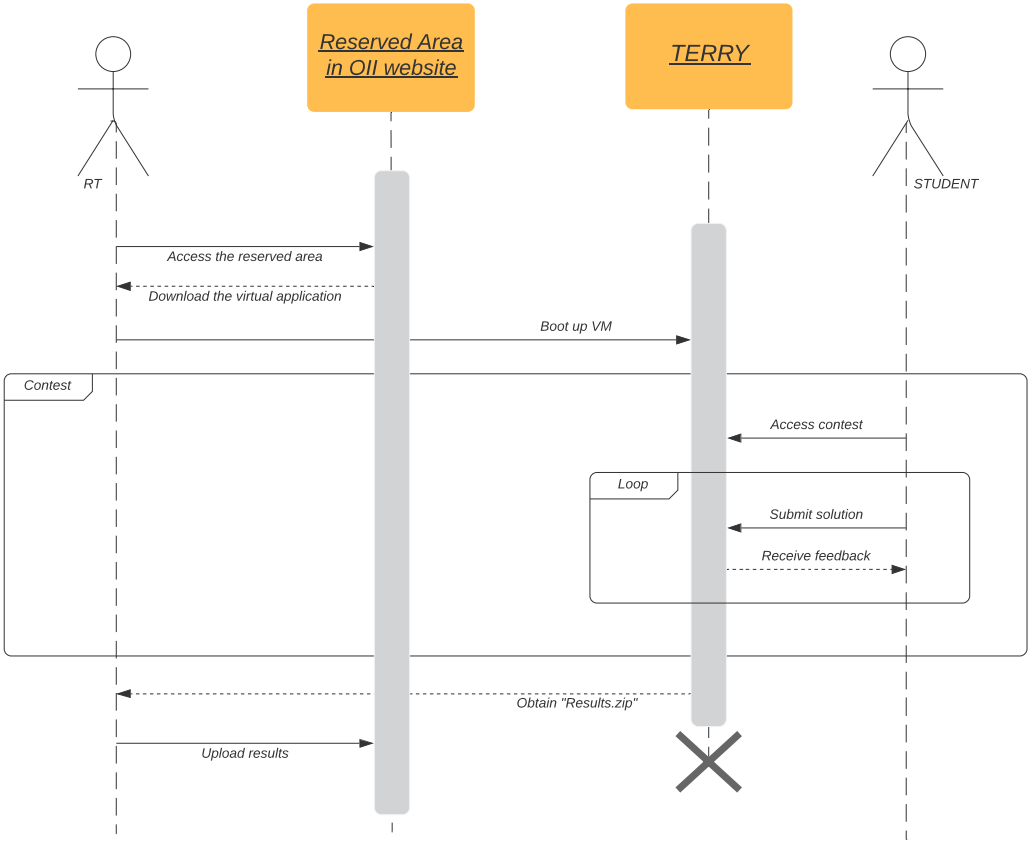


Figure 3.12: Sequence diagram showing the operations

3.3.3 Contest Admins

The OII staff, i.e. the contest administrator, will produce a virtual application where all the software dependencies of Terry are already installed. The task data such as generator scripts and model solutions will be managed on the file-system in a way that closely resembles the structure that CMS already uses (see 2.3.2) and then imported in Terry's database before packing the VM.

This VM will be then uploaded in the "reserved area" of the OII website, accessible to RTs. At the end of the contest, in the same reserved area the RTs will upload all the zip files obtained from Terry. After collecting the zip, they will be merged to form the final ranking of the contest.

If the district has a working Internet connection during the competition, the contest administrators will be able to remotely access and inspect the VM. To do this, the VM will be equipped with a set of scripts that remotely establish a tunneled connection to a single, central server.

4 Implementation of Terry

In this chapter we will see more in details how we actually implemented Terry, focusing especially on the front-end code, where the author contributed to the implementation.

4.1 Choices

Given the requirements in 3.1, we decided that Terry should be radically different from CMS in the way in handles the evaluation.

4.1.1 No-run Evaluation

Similarly to what happens during some online competitions such as the pre-2018 version of *Google Code Jam* which we already mentioned in 1.3, we will implement a system that does not require to run the users' programs in order to perform an evaluation.

More specifically, the system will generate different input files at the user's request, and it will evaluate the *correctness* of the user's solution by just checking the output file produced for said input.

This choice, even though it lets us evaluate the correctness of the program, won't let us evaluate the *optimality* in terms of time and memory, since someone can run their own program for a long time or they can use more

memory than we would otherwise allocate for them. These points, however, don't concern the district-level selection as we established in 2.6.1.

This choice allows us to forget about many of CMS's complexity such as Workers, since the evaluation step has a *predictable* and mostly very low running time and memory footprint. There's no need for a secure sandbox either, since we don't execute user's code.

4.1.2 Distribution of Terry

In order to satisfy requirements about fairness, we won't require district to have Internet access and we will distribute Terry as a virtual machine, taking inspiration from how the same contest used to be managed in the past.

4.2 Technologies

Since our team had extensive experience with CMS, it was only natural that we would tend to use some of the same technologies in order to expedite the development of the new software. In fact, we decided to use Python for all the server-side code, and for the client-side we opted to use TypeScript.

4.2.1 Python

Python is a multi-paradigm programming language. We decided to adopt it for our project because of its flexibility and for the fact that, being an interactive language, it's especially appropriate for easily testing code snippets without the need for running an entire script every time. Moreover, the Python standard library contains a wide range of tools that in other languages might be available only as third-party libraries. For example, the `json` and `csv` modules.

hello.py

```
print("Hello world!")
```

```
$ python hello.py
Hello world!
```

Listing 1: Execution of a simple “Hello world” in Python

On top of the standard library, Python puts at our disposal community-driven tools and libraries that can be installed with the `pip` package manager. To install a third-party Python package, it’s enough to run:
`$ pip install packagename`

4.2.2 Typescript

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical *superset* of JavaScript which adds optional **static typing** to the language. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs.

add.ts

```
function add(left: number, right: number): number {
  return left + right;
}
```

```
$ tsc add.ts && cat add.js
function add(left, right) {
  return left + right;
}
```

Listing 2: Transpiling TypeScript code down to JavaScript

The biggest selling point of TypeScript for us was that it helps removing a wide range of logic errors that would otherwise go unnoticed in traditional JavaScript. The simple fact that a TypeScript program *compiles* (more precisely: transpiles) is a guarantee that there cannot be errors of type mismatch in the code.

4.2.3 SQLite

SQLite is a RDBMS written entirely in C and widely used as a database system embedded directly inside other applications. It has its pro and cons, for example: it can boast the fact that the database is completely contained in a single file, but this means that it's limited when it comes to concurrent writes.

The SQL commands available are also limited (e.g. the `ALTER TABLE` commands are not supported) but this is is not a problem for our use case since Terry is not supposed to be run "indefinitely" so it doesn't require schema updates.

We chose SQLite as our database system because of its versatility and wide compatibility with all the languages and frameworks that we were already accustomed with.

```
$ sqlite3 sampledb.sqlite
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> SELECT count(*) FROM students;
26
sqlite>
```

Listing 3: Example of command line usage of SQLite

4.2.4 ReactJS

ReactJS is a JavaScript library developed and maintained by Facebook that helps with the creation of complex web applications that need to dynamically fetch and react to data changes. It is only one among many other frameworks that solve the same problem, like AngularJS, VueJS, SvelteJS and others.

The reason we chose ReactJS over the other framework was purely because we were already familiar with this particular library, so it made sense to choose it in order to avoid having to learn a different one.

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

Listing 4: Example of a React "component" that prints a greeting

In React, the application is usually split up in different components. Each component has life-cycle callbacks that lets us control what happens right after its creation, change of state, destruction and so on.

4.2.5 Bootstrap

Bootstrap is a popular CSS framework that we chose to use to build our UIs. It contains CSS-based design templates for typography, forms, buttons, navigation and other interface components.

The main reason we chose Bootstrap is because it let us quickly develop a visually appealing UI that was also accessible and (even though not required by this project) also mobile-ready.

4.3 Back-end implementation

The "back-end" component of Terry is a Python process that will act as both a web server that receives HTTP requests from the contestants' browsers, and as a dynamic queue that will handle the generation of inputs and validation of outputs, using SQLite and the file-system to store all the data about the running contest.

4.3.1 Web server

The web server will provide:

- The static HTML of the front-end;
- The compiled JavaScript and minified CSS;
- All necessary icons and images required by the front-end;
- An API that the front-end code will use to read and write information about the contest.

4.3.2 Terry API

The API that the contestant-facing web application frontend is going to consume is shown in the following figure.

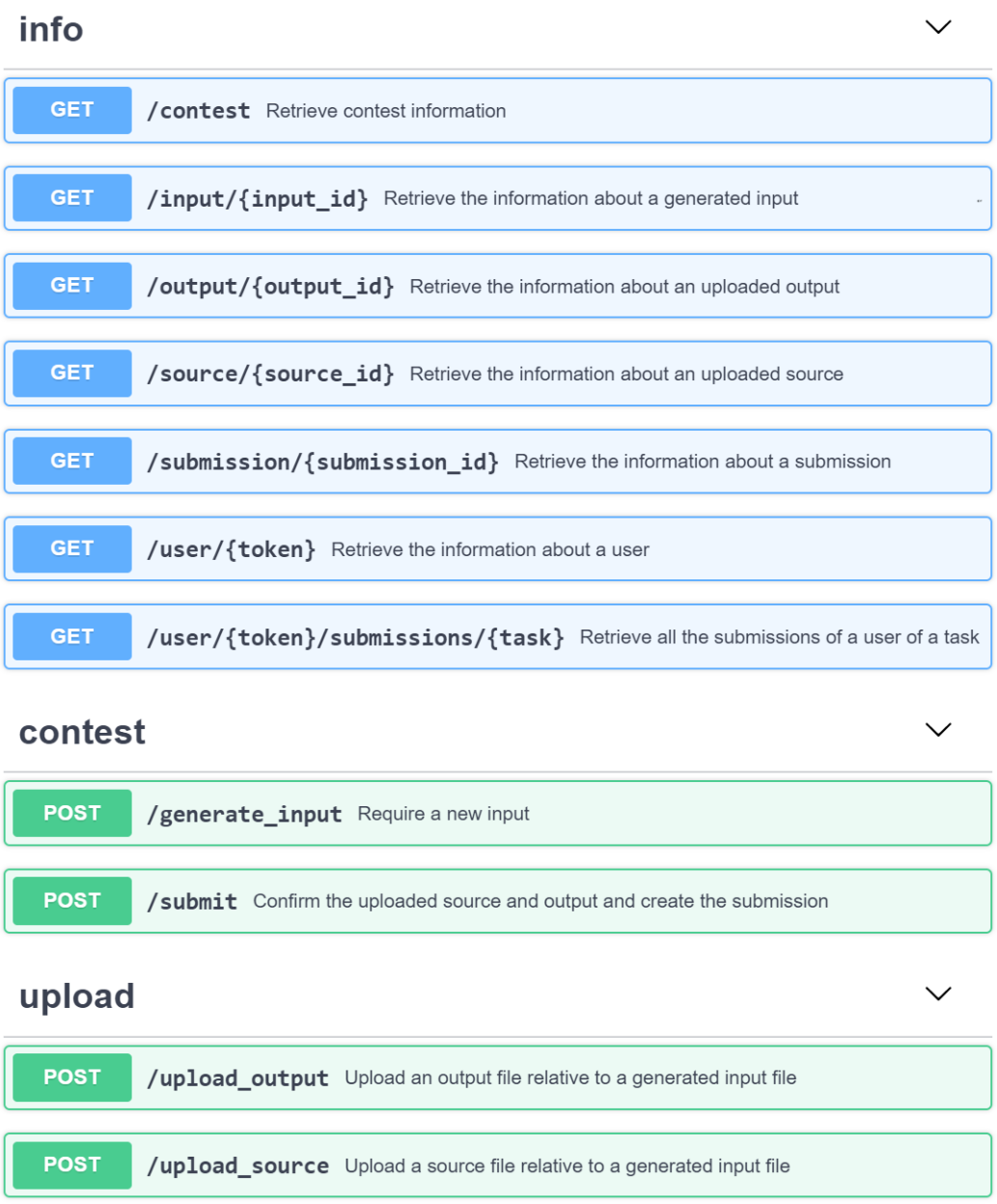


Figure 4.1: A subset of the API supported by the Terry backend.

4.3.3 Dynamic queue

As per requirements, we need to serve a different input file every time a submission is finalized. To make this possible, Terry will manage a self-replenishing pool of available inputs. The pool size is an implementation detail: we saw that 64 seem to be working fine. As soon as one input is requested, Terry will assign it to the contestant who made the request and schedule the generation of a new input as a background process.

After boot, Terry will concurrently start serving HTTP requests and generating the initial pool of input files.

4.4 Front-end implementation

The front-end application code is split in components with the main one being the `ContestView` component.

4.4.1 The `ContestView` component

This component includes a navbar at the top with user controls and a timer that periodically checks for an Internet connection which, if detected, is logged so that the contest administrators are aware of the misconfiguration of the contestant workstation and can inform the RT about it.

It is formed by two sub-components: the `SidebarView` and the `TaskView` component. The latter is dynamically reloaded when there is a change in the URL that is being visited.

In order to obtain the list of tasks to show in the sidebar, the `ContestView` connects to the backend API.

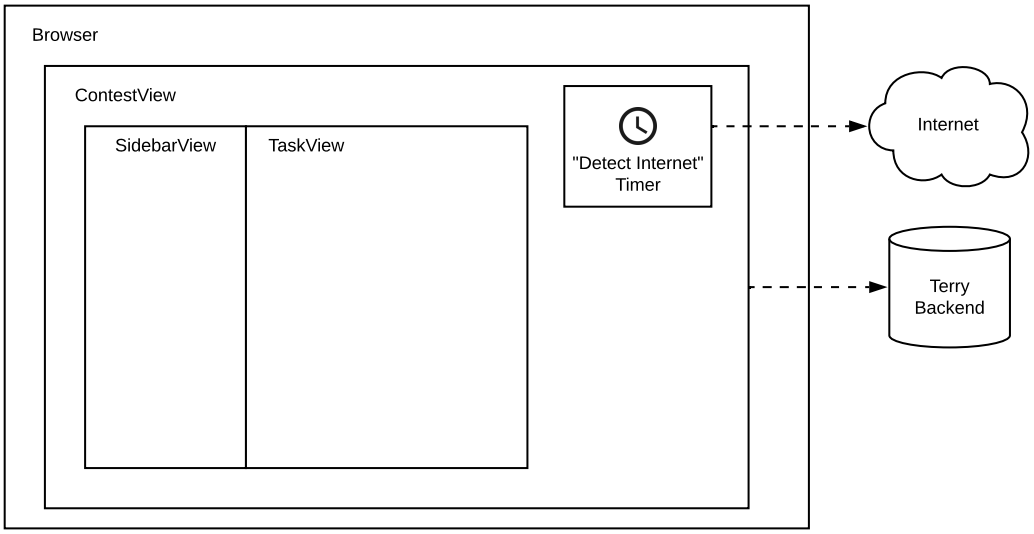


Figure 4.2: The `ContestView` component.

4.4.2 The `TaskView` component

The `TaskView` component includes the logic to choose the set of commands that are available. Based on what the contestant has already done during the contest, the `TaskView` will either show or hide some widgets.

The "Generate Input" button is visible if the user does not have an input already assigned for that task. Otherwise, the "Download Input" and "Upload Solution" buttons are shown.

The "List of submissions" link is visible if the user has already one or more submissions on that task. Otherwise, no link is shown.

Below those buttons, a `TaskStatementView` is included.

Moreover, the `TaskView` component will show a modal with other components based on the URL that is currently being visited. This is to implement the User Interface Requirements shown in 3.2 and to satisfy the requirement of Intuitive Interface described in 3.1.6.

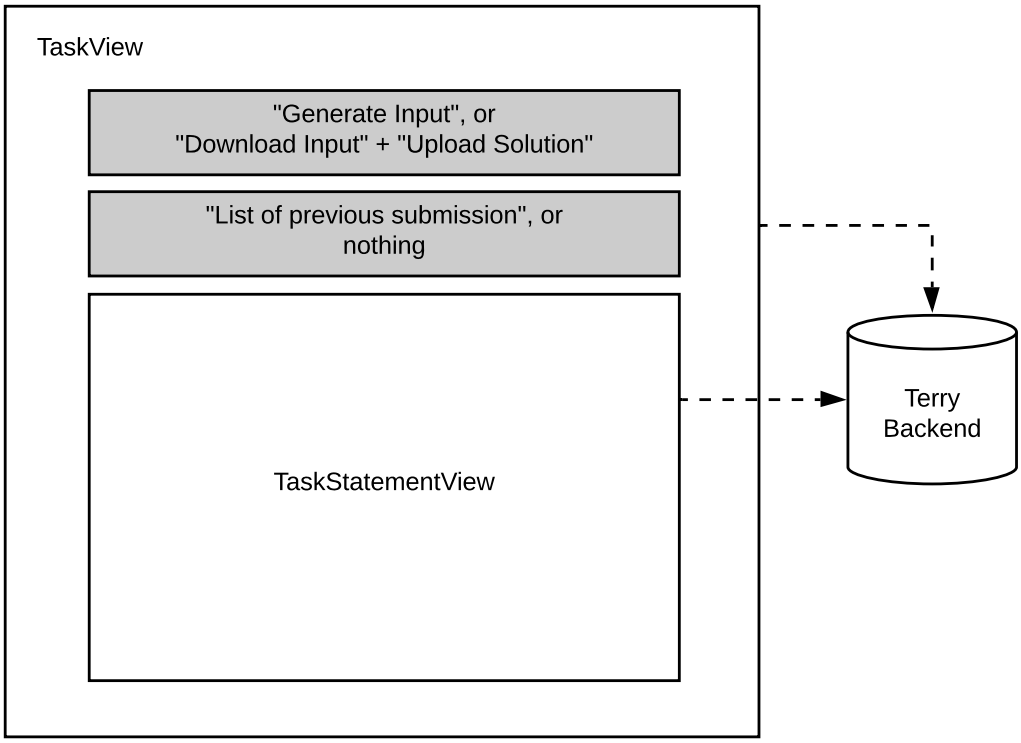


Figure 4.3: The `TaskView` component.

The choice of including the other "paths" (like the submission dialog, the list of submissions and so on) in a modal makes sense because in this way the users are always aware of how they reached that point in the application and how to come back from it.

4.4.3 The `ModalView` component

The `ModalView` is a component that helps us with "wrapping" other components inside a Bootstrap modal container.

We use this component from `TaskView` to make other components appear when the user visits some specific routes.

4.4.4 The `SubmissionView` component

The `SubmissionView` wraps inside a `ModalView` the submission form for a specific input. It effectively implements the same logic that we first described in the state diagram shown in 3.3.1.

4.4.5 The `SubmissionListView` component

The `SubmissionListView`, again wrapped inside a `ModalView`, is a component devoted to showing a sorted list of all submissions made by the contestant on the task.

4.5 Front-end screenshots

Figure 4.4: The `TaskView` as it appears before requesting any input

Figure 4.5: The `TaskView` as it appears after requesting an input

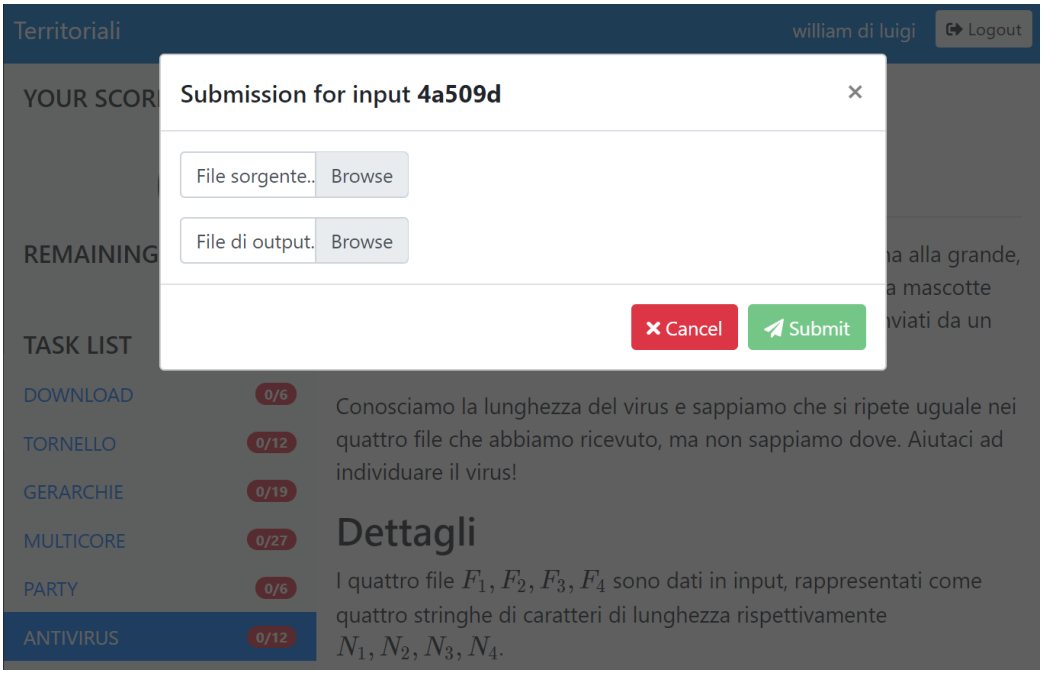


Figure 4.6: The `SubmissionView` in a modal

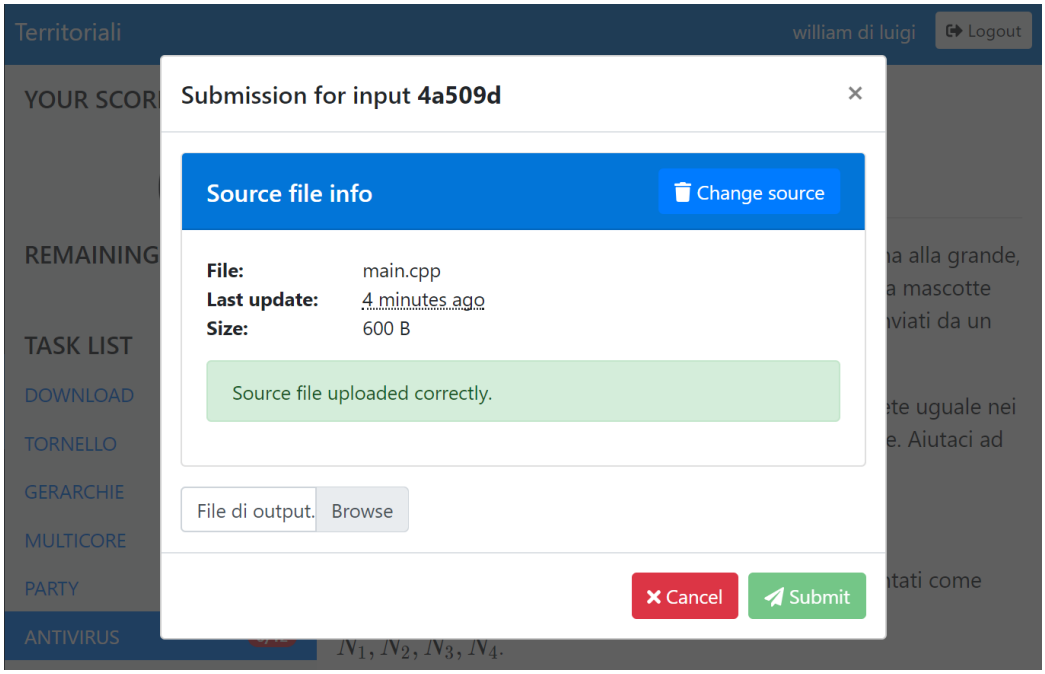


Figure 4.7: The `SubmissionView` in a modal, after uploading a source

Figure 4.8: The `SubmissionView` in a modal, after submitting

Figure 4.9: Back to `TaskView` in a modal after having submitted

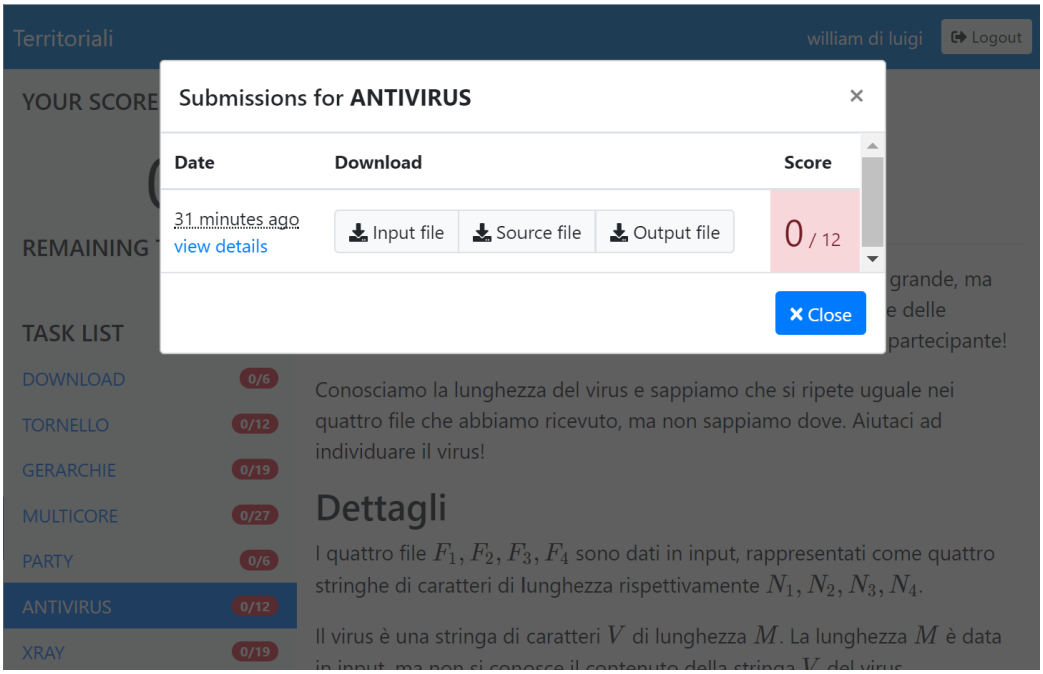


Figure 4.10: The `SubmissionListView`, now accessible from the `TaskView`

Conclusion

In this thesis we introduced the Informatics Olympiads, and the pipeline that in Italy is put in place to select the best students that should represent the country in this competition. We described the steps that this pipeline has and we focused our attention on the software and procedures used by the organizers to perform the various phases of the selection.

Having seen how CMS is not appropriate to be used in the district-level selection, we then made a list of requirements that we wanted to satisfy and we designed a new software: Terry.

The development of this software was carried out using technologies such as:

- Python
- SQLite
- TypeScript
- ReactJS
- Bootstrap

As of 2019 Terry has been used twice, both times successfully, in the district-level selection of the Italian Olympiads in Informatics.

List of Figures

Bibliography

[1] *Overview of Olympic Games*. URL: http://www.britannica.com/EBchecked/topic/428005/Olympic-Games.

[2] *International Science Olympiads*. URL: http://olympiads.win.tue.nl/.

[3] *International Olympiads in Informatics*. URL: https://ioinformatics.org/.

[4] Christian R. Prause and Zoya Durdik. "Architectural design and documentation: Waste in agile development?" In: *International Conference on Software and System Process (ICSSP)* (2012).

[5] Borja Sotomayor Aaron Bloomfield. "A Programming Contest Strategy Guide". In: *SIGCSE* (2016).

[6] *Final Report International Olympiad in Informatics 1992 Bonn / Germany*. URL: http://olympiads.win.tue.nl/ioi/ioi92/report.html.

[7] Rosalie Chan. "The 10 most popular programming languages, according to the Microsoft-owned GitHub". In: (2019).

[8] *Compiled versus interpreted languages*. 1990. URL: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zappldev/zappldev_85.htm.

[9] Wouter T. van Leeuwen. “A critical analysis of the IOI grading process with an application of algorithm taxonomies”. In: (2005). URL: https://pure.tue.nl/ws/portalfiles/portal/47048826/599683-1.pdf.

[10] *The Sixth International Olympiad in Informatics: A Trip Report, 3 - 10 July 1994, Haninge, Sweden*. URL: http://olympiads.win.tue.nl/ioi/ioi94/rprt-nl/index.html#Judging_the_Programs.

[11] *DOMjudge at the ICPC World Finals*. 2012. URL: https://www.domjudge.org/pipermail/domjudge-devel/2012-May/000922.html.

[12] G. Mascellani S. Maggiolo. “Introducing CMS: A Contest Management System”. In: *IOI Journal* 6 (2012), pp. 86–99. URL: https://www.mii.lt/olympiads_in_informatics/files/volume6.pdf#page=89.

[13] G. Mascellani S. Maggiolo. “CMS: a growing grading system”. In: *IOI Journal* 8 (2014), pp. 123–131. URL: http://www.mii.lt/olympiads_in_informatics/files/volume8.pdf#page=125.

[14] *La storia delle OII*. Italian. URL: https://olimpiadi-informatica.it/index.php/oii/la-storia-delle-oii.html.

[15] *Italian Team Olympiads in Informatics*. URL: https://squadre.olinfo.it/about.

[16] *The Psychology of choice: Why less is more*. URL: https://www.keepitusable.com/blog/the-psychology-of-choice-why-less-is-more/.