Development of an automated system for conducting, checking and evaluating programming competitions

Bohdan V. Hrebeniuk, Olena H. Rybalchenko

Kryvyi National University, 11 Vitalii Matusevych Str., Kryvyi Rih, 50027, Ukraine

Abstract

The paper analyzes the existing platforms for conducting programming contests. Possible approaches are analyzed for creating isolated environments and running participants' solutions, advantages and disadvantages of both approaches are highlighted. Requirements for the user interface are defined that must provide quick and convenient work in the system; the system was planned and developed. It was concluded that designed system has a potential for conducting contests and further development.

Keywords

competiteve programming, isolated environment, virtualization, containerization, REST

1. Introduction

Competitive programming has been maintaining a dynamic development rate for several decades. Year by year, more and more teams take part in local, regional and international contests, therefore making the needs of both contest organizers and participants grow.

Kryvyi Rih National University supports the development of competitive programming – local contests are held, and the university annually hosts the qualification phase of the international student contest under the aegis of ACM/ICPC. To maintain the level of students and teachers competence, it is necessary to have our own flexible system, since existing analogues have certain disadvantages.

The aim of the work is to study and analyze the process of creating isolated environments, design and develop a flexible system for conducting, checking and evaluating programming contests.

CS&SE@SW 2020: 3rd Workshop for Young Scientists in Computer Science & Software Engineering, November 27, 2020, Kryvyi Rih, Ukraine

bogdan020699@gmail.com (B.V. Hrebeniuk); rybalchenko@knu.edu.ua (O.H. Rybalchenko)
0000-0002-0423-8476 (B.V. Hrebeniuk); 0000-0001-8691-5401 (O.H. Rybalchenko)

© 2020 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-2832/paper14.pdf

2. Development of a system for conducting, checking and evaluating programming competitions

2.1. Analysis of existing systems

One of the first systems with automatic checking of tasks is UVa Online Judge, an online system of the University of Valladolid with an installed task archive. The archive is constantly being updated, but it is impossible to create tasks yourself using this system – any system updates (uploading new tasks, supporting new programming languages, changing the rules for creating a rating) are possible only with permission and active interaction with developers. Additionally, downloading the software and installing it for use are prohibited due to the product license, because the system uses the "SaaS" (Software as a Service) model [1]. Under this model, the software is not installed on users' computers, but on the provider's servers, and the user does not have any access to the system. This disadvantage makes the system unsuitable for long-term contests due to the fact that the terms of the service provider can change at any moment and the user (in our case, the organizer of the contest) will not be able to do anything [2].

One of the most successful systems in history was PCSM2, which was formed by combining the APPES and PCMS systems and is characterized by the flexibility of the internal structure. Software units have low connectivity due to the fact that they rely on high-level abstractions of each other. This makes the PCSM2 system easy to upgrade on a programming level. But this flexibility has made it difficult to customize the server, as most of the business logic is described by configuration files. Some parts of the system are still being developed, so to set them up, you need to add and connect specialized modules that only the authors of the system can perform. Another serious drawback is that the PCSM2 server supports only the Windows operating system, which makes it difficult to run the system on servers that use Linux [3].

The qualification phases of the ACM/ICPC contests are managed by the ejudge system [4], which is significantly different from PCSM2. The system was created using the C programming language, it supports only the Linux operating system, has a web interface for administration and detailed documentation, but also has a number of disadvantages. The most important of these is the inefficient allocation of resources between the system and processes with user's solutions. As a result, incorrect verdicts may occur due to exceeding the time limit. The system also doesn't support scaling out — you can't connect a new machine to an existing one. The only way to improve parallel code execution is to increase the number of processes on the server, which can lead to competition for process access to RAM. An equally important factor is linking the system to the operating system. This dependency makes it impossible to deploy the project on servers that do not support Linux.

2.2. Analysis of approaches of running participants' code

Automation of contest processes has become a necessary condition for such events, because with the increase in the number of teams and tasks, the required amount of time for receiving solutions, checking them, and summing up results has also significantly increased. And, although such systems have existed for decades, with the introduction of new technologies, new opportunities for their creation appear.

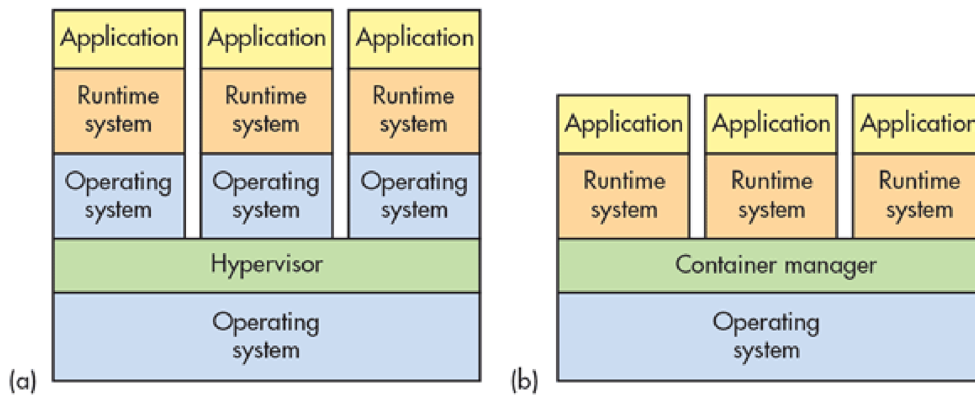

Figure 1: Component hierarchy for virtualization (a) and containerization (b) approaches

To protect such systems from malicious user code actions, the common practice is to run the code in some isolated environment. This makes the system protected from malicious solutions that may attempt to interfere with the evaluation process [5].

One potential solution to this problem can be virtual machines that emulate a hardware server. In this event, all components of a real computer are virtualized – I/O devices, hard disk, ports, etc., followed by the installation of the operating system. Although such system is isolated from the host, using virtual machines means running the entire operating system and allocating resources to emulate virtualization, which often negatively affects the performance of the platform.

An alternative way to achieve process isolation is containerization, in which the source code of the program is encapsulated along with dependencies to run on any infrastructure in an isolated environment [6]. Unlike virtual machines, containers allow you to run software in a single process, isolated from each other, on a single host, using the resources of an existing operating system. Since containers are only a part of the operating system, they are sometimes less flexible to use, for example, they can only run programs that are compatible with the current system [7].

Despite mentioned limitations, containers are widely used to create an isolated process environment. One of the most well-known examples of such containers is “chroot jail”. The idea is to copy (or create) links to the system files needed to run processes. After that, restrictions are set for the created processes — the root directory of the system is changed to the root directory of the environment. Since processes cannot refer to paths outside the modified root, they cannot perform malicious operations in these locations [8]. Containerization is the best option for the system being created, because containers allow you to run programs in an isolated environment with restrictions on system resources and at the same time work only in one process of the operating system.

2.3. Docker usage

One of the tools for working with containerization is Docker, which allows you to create an isolated space for a process using the Linux kernel namespace. Each container creates its own spaces with unique access, including the following:

- PID namespace (for process isolation);
- NET namespace (for managing network interfaces);
- IPC namespace (for managing access to IPC resources);
- MNT namespace (for managing file system mounting points), etc.

Docker also uses kernel control groups to allocate resources and isolate them. These groups set limits for the program for certain resources, including:

- memory control groups;
- CPU control groups;
- control groups of devices, and so on.

The containers must be used in our platform for the next purposes:

- compiling the user's code;
- running the user's code.

Therefore, it is necessary to have two containers for each programming language in the system – one container compiles the code, the second executes it. Creating containers is achieved by running images, which in turn encapsulate all the components and dependencies needed to run the application. Images are created using Dockerfile - text files with instructions created according to the rules of a Dockerfile syntax.

For example, let's take a DockerfileBase file for creating images for the Python programming language:

```
FROM python:3.7.5-slim
COPY compile_py.py .
```

This file describes the image that will be based on the python:3.7.5-slim image, the official Python version 3.7.5 image, the slim build. Copy the script to the virtual address space compile.py, which is responsible for compiling the user's code. The contents of the file are shown below:

```
import pathlib
import py_compile
import sys
```

```
if len(sys.argv) != 3:
    raise ValueError('Wrong amount of arguments!')
_, input_file, output_file = sys.argv
input_file_path = pathlib.Path(input_file)
if not input_file_path.exists():
    raise ValueError('There is no user file!')
py_compile.compile(input_file, output_file, doraise=True)
```

The base image (python-base) is created before the platform starts working. Based on this image, we create two new images – DockerfileCompiler and DockerfileRunner for compiling and running code respectively. The content of DockerfileCompiler is shown below:

```
FROM python-base
COPY ./code /code
CMD ["python", "compile_py.py", "./code/main.py", "./code-compiled/main.pyc"]
```

The content of the DockerfileRunner file is as follows:

```
FROM python-base
COPY ./code-compiled /code-compiled
CMD ["python", "/code-compiled/main.pyc"]
```

For each new user solution, a pair of such images will be created. An image based on DockerfileCompiler copies the code directory to its virtual address space and runs the script compile.py, which compiles the source code to byte-code. Due to the presence of volume (which must be configured before starting the container), the contents of the directory will remain available to other processes, in particular for the image created from the DockerfileRunner file, on the basis of which the container will execute the compiled code.

To simplify interaction with the Docker Engine API, which is an interface in the form of an HTTP client for other applications, a facade is implemented that is a wrapper over the client. Its main task is to define the interface of a real client by transferring the necessary methods to a single class:

- build-creates an image based on a specific Docker file and assigns it a name;
- run-starts a container with a specific configuration based on the specified image and returns the result of its operation;
- remove-deletes a specific image.

This class is used in "workers" – components that encapsulate the logic of working with containers and images for a given set of programming languages. Workers are waiting for the user's solution, which is used to create a context for running code, namely:

- creating a directory with further configuring volumes;

- building a “Compiler” – an image and creating a one-time container based on it;
- building a “Runner” image based on a compiled solution.

To avoid creating new “Runner” images to run the solution with different tests, input data is passed in arguments to run the container. After testing the solution, the docker context is destroyed (images and containers are deleted).

After implementing a software component to run user solutions, the next step is to create a system that meets business requirements.

2.4. System’s architecture

The client-server architecture was chosen as the basis for the created software, so the selection of frameworks for developing both the backend and the frontend turned out to be an important issue.

The Python programming language (specifically version 3.7.4) was used to develop the backend. Although Django and Flask are the leading frameworks for creating Python web applications, a relatively new aiohttp framework was chosen to develop our system. The framework uses new features of the language, namely support for asynchronous programming at the syntax level [9]. The main concept of this paradigm is the use of deferred calls, which violate the usual order of instructions execution. At the same time, the resource usage model significantly differs from the model using processes and threads – there is only one thread in an asynchronous application, and new requests are served using asynchronous I/O primitives of the operating system – as a result, a small amount of resources is used for allocating new connection. Although the framework was created relatively recently (the first official “stable” release dates back to September 16, 2016), 3 major updates were released in this short period of time, and the framework repository on GitHub became the “main” one in AIO-libs, a community of programmers who develop Python packages and modules for asynchronous programming [10].

The choice of such a framework affected the choice of database, because there is only one adapter written for Python that supports asynchronous database access - that is psycopg2, a connector for PostgreSQL. Based on this adapter, the aiopg library package was created, which implements a DBAPI interface with asynchronous syntax support and is responsible for supporting SQLAlchemy, a library for working with relational databases using ORM Technology [11].

It is worth noting that not only PostgreSQL implements support for asynchronous operations. The MySQL DBMS also has it, but there is no official client for the Python programming language that would support this mode, so the PostgreSQL was chosen [12].

The React framework was chosen for the frontend. The programming code created using this framework is modular, i.e. it consists of loosely coupled components that can be reused in several places at once. Thanks to the virtual DOM, applications created using React run much faster than alternative frameworks (for example, Angular). The Redux library, a platform with unidirectional data flow and centralized data storage, was used to manage the application status. These features of the platform allow you to implement a predictable and easy-to-understand system for moving an application from one state to another.

REST was chosen as the architectural style of interaction between the backend and the frontend. Some features of this architecture are described below [13]:

- client-server architecture – separating the interface from business logic simplifies the portability of the user interface to different platforms and increases scalability by simplifying server components;
- state absence – each request to the server must contain all the necessary information for its successful execution, i.e. the server does not store any context between different requests (for example, in the form of sessions), and all information needed for the user's "session" is stored on the client's side (for example, an access token);
- uniform interface – following REST involves creating a "virtual" resource system, in which each of the resources has universal methods for working with it, including creating, reading, updating, and deleting. However, the true location of the entity (for example, in a database) may be located in a different virtual space.

The functional diagram shows a graph with transitions from one state of the program to another. When initializing the system, connections to the Docker Engine API are created; connecting to the database and starting the web server. After that, the system waits for a request from the user.

Each request to the system's API is accompanied by user authorization – it checks whether the user can have access to the specified resource. If authorization fails, the user is sent a message describing the cause of the error. In other case, the request goes further through the system.

There are two main types of API requests:

- a "CRUD request" is a request for manipulating an entity that requires interaction with the database;
- a code execution request is a request to execute code from the user in order to get the result of the program. The request requires interaction with the Docker Engine API.

2.5. User interface and user experience

One of the necessary requirements for the system is the speed and convenience of creating and conducting competitions, so certain restrictions were set on the system interface for effective user experience.

The most important of them is that the user should not be focusing on the elements that are not related to the context of the current resource - it is necessary to create a virtual space that does not allow the user to move randomly around the system, but on the contrary, suggests the right actions to achieve the goal.

For example, the pages "view available competitions" and "create a new competition" belong to the same type of system resources - "competitions", so manipulations on this resource are "close" to each other in the virtual space of the system.

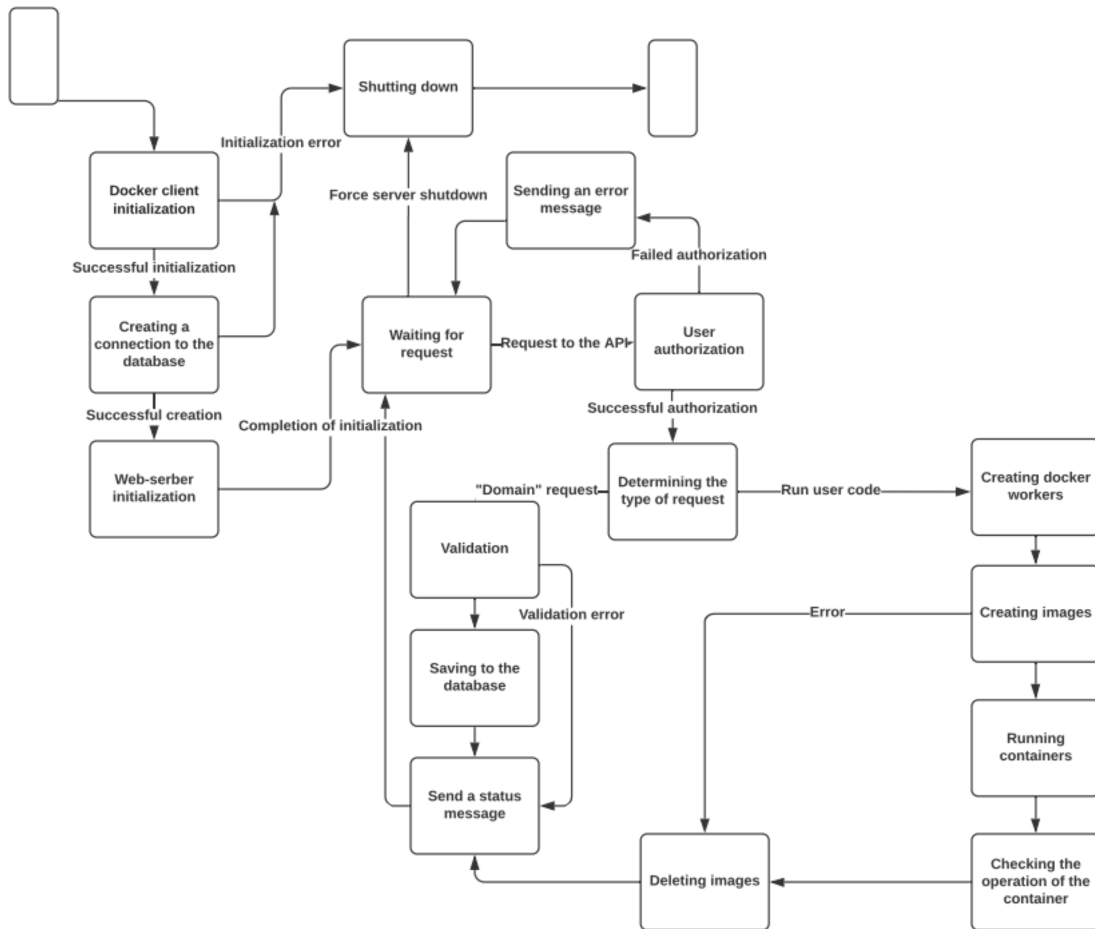

Figure 2: Functional diagram of the system

When viewing a specific competition, the user has more features, so the header has significantly more elements – the virtual space has expanded significantly at this stage, and links to loosely linked pages (creating a new competition) are not displayed.

The user experience of the system is based on building clear sequences of actions that lead to resources in one most intuitive way. This design allows you to focus the user's attention on performing the planned action, instead of confusing them with oversaturated pages and long transitions.

3. Conclusions

Research on technologies for creating isolated environments has shown that containerization, despite a number of disadvantages, is well suited for running processes isolated from each other. The use of virtual machines, on the contrary, could lead to irrational use of system resources, so it was decided to use containers as components for running user code.

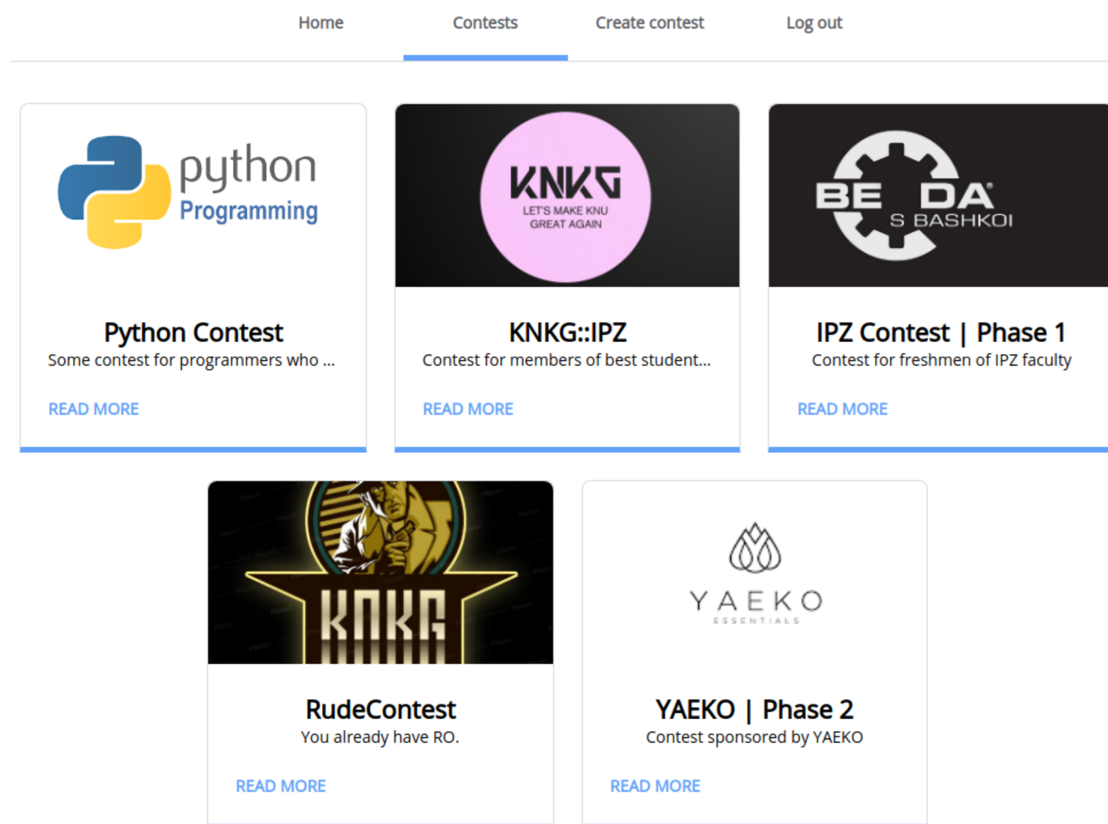

Figure 3: Contest list view page

After the development of this component, a system for holding contests was created, which can be used by both participants and contests organizers. A scheme for a relational database is designed for the system. The software is designed to emulate the client/server architecture, using Python and JavaScript. The system has passed several tests, which have shown an effective system operation and a great potential for further development.

References

[1] O. M. Markova, S. O. Semerikov, A. M. Striuk, The cloud technologies of learning: Origin, Information Technologies and Learning Tools 46 (2015) 29–44. URL: https://journal.iitta.gov.ua/index.php/itlt/article/view/1234. doi:10.33407/itlt.v46i2.1234.

[2] D. Irtegov, T. Nesterenko, T. Churina, Systems for automated evaluation of programming tasks: Development, use and perspectives, Vestnik NSU. Series: Information Technologies 17 (2019) 61–73. doi:10.25205/1818-7900-2019-17-2-61-7.

[3] D. Irtegov, T. Nesterenko, T. Churina, Development of automated evaluation systems for programming tasks, System Informatics Journal (2017) 91–116. doi:10.31144/si.2307-6410.2017.n11.

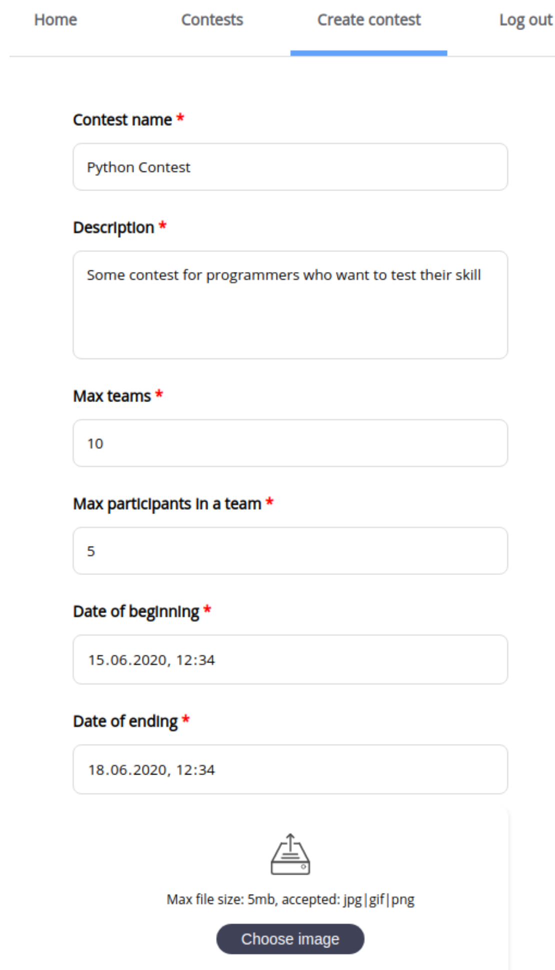

Figure 4: Contest creation page

[4] ejudge system, 2020. URL: https://ejudge.ru/wiki/index.php/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_ejudge.

[5] Selected papers ofthe International Conference joint withthe XXII International Olympiad in InformaticsWaterloo, Canada, August 14–21, 2010, volume 4, Institute of Mathematics and Informatics, Vilnius, Lithuania, 2010. URL: https://ioinformatics.org/files/volume4.pdf.

[6] IBM Cloud Education, Containerization explained, 2019. URL: https://www.ibm.com/cloud/learn/containerization.

[7] J. Turnbull, The Docker Book: Containerization Is the New Virtualization, 2014. URL: https://dockerbook.com/.

[8] chroot "jail" - what is it and how do I use it?, 2010. URL: https://unix.stackexchange.com/questions/105/chroot-jail-what-is-it-and-how-do-i-use-it.

[9] Y. Selivanov, PEP 492 – Coroutines with async and await syntax, 2015. URL: https://www.

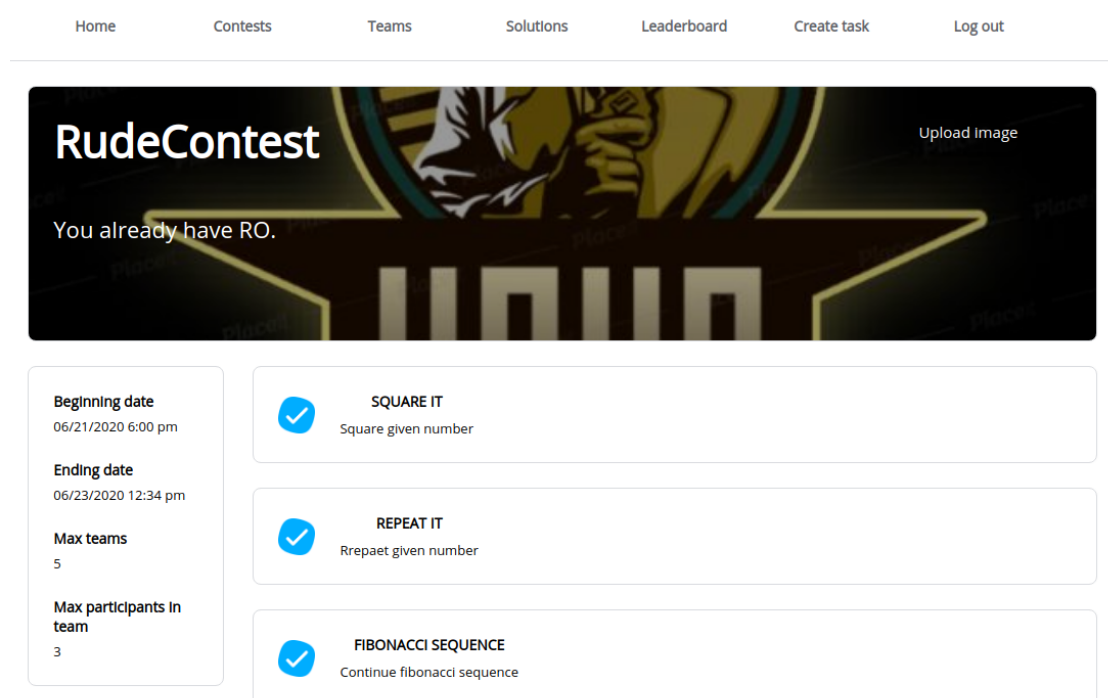

Figure 5: Contest resources overview page

python.org/dev/peps/pep-0492/.

[10] aio-libs: The set of asyncio-based libraries built with high quality, 2021. URL: https://github.com/aio-libs.

[11] A. Svetlov, A. Firsov, Welcome to AIOPG, 2020. URL: https://aiopg.readthedocs.io/en/stable/.

[12] Is there an asynchronous Python client for MySQL with pooling features?, 2015. URL: https://www.quora.com/Is-there-an-asynchronous-Python-client-for-MySQL-with-pooling-features.

[13] restfulapi.net, What is REST, 2020. URL: https://restfulapi.net/.