IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/45SE/2025/4539225/
Vol. 2025 (22), Article ID 4539225

An Integrated System for Managing and Evaluating Student Code Submissions*

Tomasz GUTOWSKI

Military University of Technology, Warsaw, Poland

Correspondence should be addressed to: Tomasz GUTOWSKI, tomasz.gutowski@wat.edu.pl

* Presented at the 45th IBIMA International Conference, 25-26 June 2025, Cordoba, Spain

Abstract

This paper presents the development and deployment of a web-based platform designed to improve the process of managing and evaluating programming assignments in academic setting. The system integrates three components: assignment creation, automated grading, and code similarity detection — all within a unified, user-friendly interface. Its primary objective is to reduce the time and effort required by teachers to assess large volumes of student submissions while maintaining fairness, consistency, and transparency in grading. The platform allows teachers to define assignments and corresponding test cases, which are then used for automatic evaluation of submitted code. The grading engine executes student programs in a secure environment and compares the outputs against expected results, providing detailed feedback for each test case. In addition, the system features a built-in similarity detection module that highlights pairs of potentially plagiarized solutions. This mechanism supports teachers in identifying dishonest practices while preserving students' privacy and autonomy. The system was successfully implemented and tested in real university courses, where it demonstrated its practical value in improving both the efficiency of course management and the quality of student feedback. Teachers benefited from significantly reduced manual workload, and students gained faster access to consistent evaluations. While the platform currently supports a limited set of programming languages and uses a relatively simple similarity detection approach, it lays a solid foundation for further improvement. Overall, the developed solution contributes to modernizing programming education and addressing challenges in large-scale course delivery.

Keywords: Automated Grading, Code Similarity Detection, Programming Education, Education Technology

Introduction

The demand for the development of information systems that support education has remained high for many years. This need has significantly increased during the COVID-19 pandemic, when many educational institutions, such as schools and universities, were forced to close and unable to organize classes on-site (Lalani et al., 2025; Oliveira et al., 2021). This global event has caused a rapid development in this area of research resulting in a wide range of solutions designed to aid the process of teaching and learning.

Cite this Article as: Tomasz GUTOWSKI, Vol. 2025 (22) "An Integrated System for Managing and Evaluating Student Code Submissions " Communications of International Proceedings, Vol. 2025 (22), Article ID 4539225, https://doi.org/10.5171/2025.4539225

The pandemic accelerated the adoption and evolution of software platforms that support remote communication, such as Microsoft Teams, Zoom, Google Meet and Discord. These tools were widely used by educational institutions to conduct lessons, providing synchronous interactions through voice and video streaming (Oliveira et al., 2021). While it provided crucial support, their effectiveness required the simultaneous presence of teachers and students.

Another type of solutions designed to facilitate learning are Learning Management Systems (LMS). These include software such as Moodle, Blackboard and Canvas (Dobre, 2015). These allow asynchronous learning. The teachers can publish and distribute learning materials, as well as organize grading and communication with students, without requiring real-time interaction. Moodle, which is open source, has been adopted at many schools and universities, providing many ways of customization through plugins. While these solutions facilitate the workflow between teachers and students, they do not seem to provide solutions for automated verification and grading of students' assignments.

Grading students' assignments is a time-consuming task for teachers, leaving limited opportunity for higher-impact activities such as communication with students, providing personalized feedback or supporting struggling students with specific topics (Randall & Engelhard, 2010). Automating parts of the evaluation process can provide significant improvements to the efficiency of teaching. While it is challenging to design a universal solution capable of evaluation all kinds of student work, effective domain-specific solutions can be created and deployed in education.

This paper proposes a custom-built solution dedicated for helping in the evaluation process of programming assignments in higher education. The system is dedicated to support computer science courses, especially the beginning assignments, when the students have to learn to implement specific algorithms, find solutions to programming challenges. The presented e-learning platform is a solution to facilitate the process of grading the coding assignments, which verifies the correctness of the assignment, but also finds similar works in order to reduce the chance of plagiarism.

The described platform is deployed at the university and is used in the teaching of multiple programming related courses, such as Algorithms and Data Structures, Object-Oriented Programming, and IoT Systems.

Literature Review

Over the recent years automation of student assessment has become an increasingly important research area in computer science education. With increasing class sizes and a greater emphasis on personalized learning, teachers are seeking tools that reduce manual grading time and provide timely, consistent feedback to students, which not only provide valuable feedback, but also verify if the work has been completed individually by the student. This section provides an overview of existing e-learning platforms that support automated code evaluation and plagiarism detection.

Some of the LMS have developed plugins that support automatic grading of programming exercises. One example is CodeRunner (Lobb & Harlow, 2016), a plugin for Moodle. It enables automatic grading of programming questions by executing student code within a secure sandbox environment. It supports a wide range of programming languages, such as Python, C, JavaScript, PHP and MATLAB. It can be integrated with Moodle's quiz system, providing flexibility and ease of use.

Similarly, CodEval (Agrawal et al., 2022) has been developed. It is a tool for created for another LMS – Canvas. It uses docker containers to compile and run code written by students, as well as the tests. This makes sure that the student's code is isolated from the main system running the test. The grading criteria are flexible and are tailored for early programming education, providing immediate feedback, which is critical for learning.

Codio(*Codio | The Hands-On Platform for Computing & Tech Skills Education*, n.d.) offers a cloud-based IDE along with automated grading features such as unit tests, custom test scripts, and code quality checks. It supports multiple programming languages and allows teachers to configure assignments with detailed rubrics, thus improving the overall assessment process.

These three systems have demonstrated significant benefits in reducing the needed time to grade the assignments and providing quick and almost immediate feedback to students, they improve the learning experience. However,

many of them require complex integrations with existing LMS platforms or are limited in their flexibility and language support.

One of the major challenges faced by educational institutions is the issue of non-independent work during the completion of assessed assignments. Especially in programming courses, students may be tempted to share solutions or copy code from external sources, which undermines the learning process and compromises academic integrity. This not only affects the fairness of assessment but also leads to incorrect evaluation of students' skills and knowledge. Manual comparing of solutions is prone to omissions and takes significant time. Verifying if students complete their assignments individually is especially important at the beginning, to make sure they do not skip learning the basics that are necessary to understand more complex topics.

To solve this problem multiple solutions have been developed, with the most widely used being MOSS (Measure of Software Similarity) (Bowyer & Hall, 1999). It is a tool for detecting plagiarism in code developed at Stanford University. MOSS uses token-based string matching and syntax analysis to detect structural similarities in student submissions, even when variable names and comments have been altered.

JPlag is another popular tool used in academia (Sağlam et al., 2024). It performs pairwise comparisons of code submissions to identify similarities and supports several programming languages including Java, C++, Python, and C#. JPlag presents its results in a visual format that is helpful for teachers when investigating suspected cases.

Additionally, there are plugins such as plagiarism_programming by Thanh Tri (*GitHub - Thanhtri/Plagiarism_programming: A Plugin That Integrate MOSS and JPlag to Moodle and Provide Other Useful Features*, n.d.), which integrate tools like MOSS and JPlag directly into Moodle. These allow educators to manage and analyze plagiarism detection results within the same platform where assignments are distributed and graded.

While the described tools provide valuable functionalities in both automated grading and plagiarism detection, they often operate independently of each other. Many of these are standalone platforms or require manual integration with existing LMSs, which can create technical overhead and reduce accessibility for less technically experienced teachers. Furthermore, some systems focus on a limited range of programming languages or assignment types, which may not cover the scope of programming tasks present in early computer science education.

The approach presented in this paper aims to fill this gap by offering an integrated, web-based platform that automates both grading and plagiarism detection for programming assignments. The system is designed to be flexible, modular, and extensible, allowing teachers to evaluate code, detect similarities, and manage results using a friendly interface.

System Architecture and Functionality

The platform has been designed as a responsive web app, consisting of three main components:

- Frontend – providing the graphical user interface for both students and teachers,
- Backend – providing the business logic for the system,
- Database – to store data regarding the users, the assignments and submissions,

as presented in Fig 1.

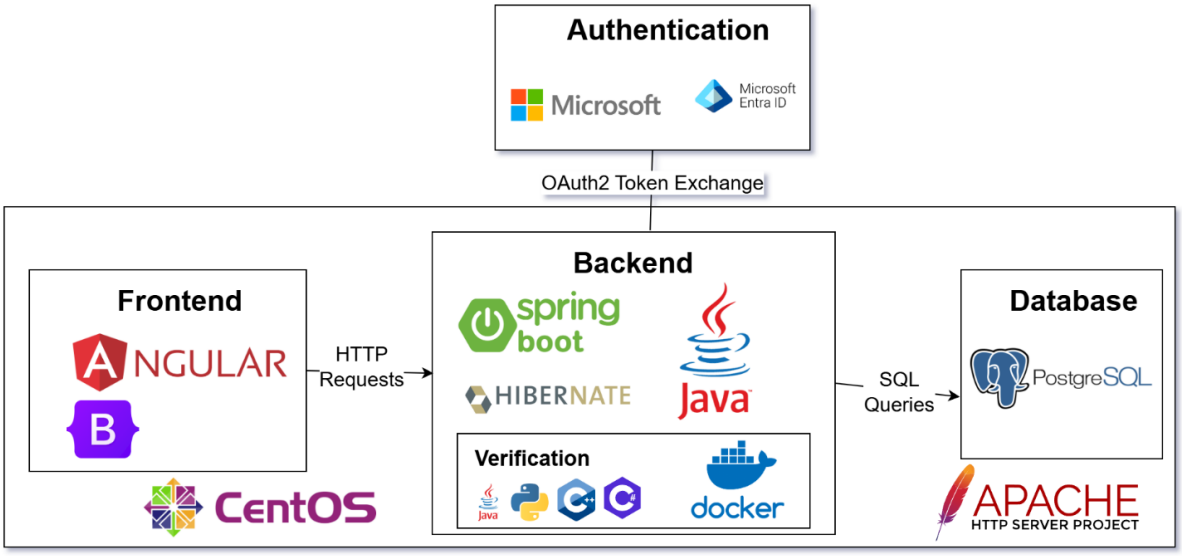

Fig 1. System architecture showing interactions between frontend, backend, and database components

The Frontend component is implemented using Angular, which is a framework designed to create Single Page Applications using JavaScript and Typescript. It provides a responsive layout and consistent styling using Bootstrap, providing a user-friendly interface accessible from various devices. The frontend handles student interactions such as assignment submissions, viewing grades and provides support for teacher activities such as creating lessons, assignments, student groups and viewing submissions.

The Backend is developed using Spring Boot with Java, providing the core business logic of the platform. It processes incoming requests from the frontend, performs assignment evaluation, manages plagiarism detection and handles authentication integration via Microsoft Entra ID. Hibernate is used as an ORM to allow communication with the database. The backend runs on a runs on a CentOS machine using the Apache HTTP server. To test students' submissions Docker is used, individual containers are started to compile and test solutions developed in various programming languages.

The Database component uses PostgreSQL for storage of platform data, including user accounts, assignments definitions, evaluation results, and submission reports similarities. The code submitted by the students is stored directly in the file system, with the paths saved in the database.

Platform Capabilities

The system is designed to provide teachers with a streamlined solution for collecting, reviewing, and grading students' programming assignments. It supports a range of features that simplify class and assignment management, these include student and teacher functionalities.

In this paper, a class realization refers to a specific combination of a subject (e.g., "Algorithms and Data Structures"), semester (e.g., "Winter 2024"), and student group (e.g., "Group A"). This term is used to describe the context in which lessons and assignments are created and managed by the teacher. Students are assigned to a student group, which in turn is associated with one or more class realizations.

Student functionalities:

- Register in student groups,
- View assigned tasks and their descriptions (as presented in Fig 2),
- Submit programming assignments (as presented in Fig 3),
- Viewing statistics (as presented in Fig 4),
- Access and review grading results.

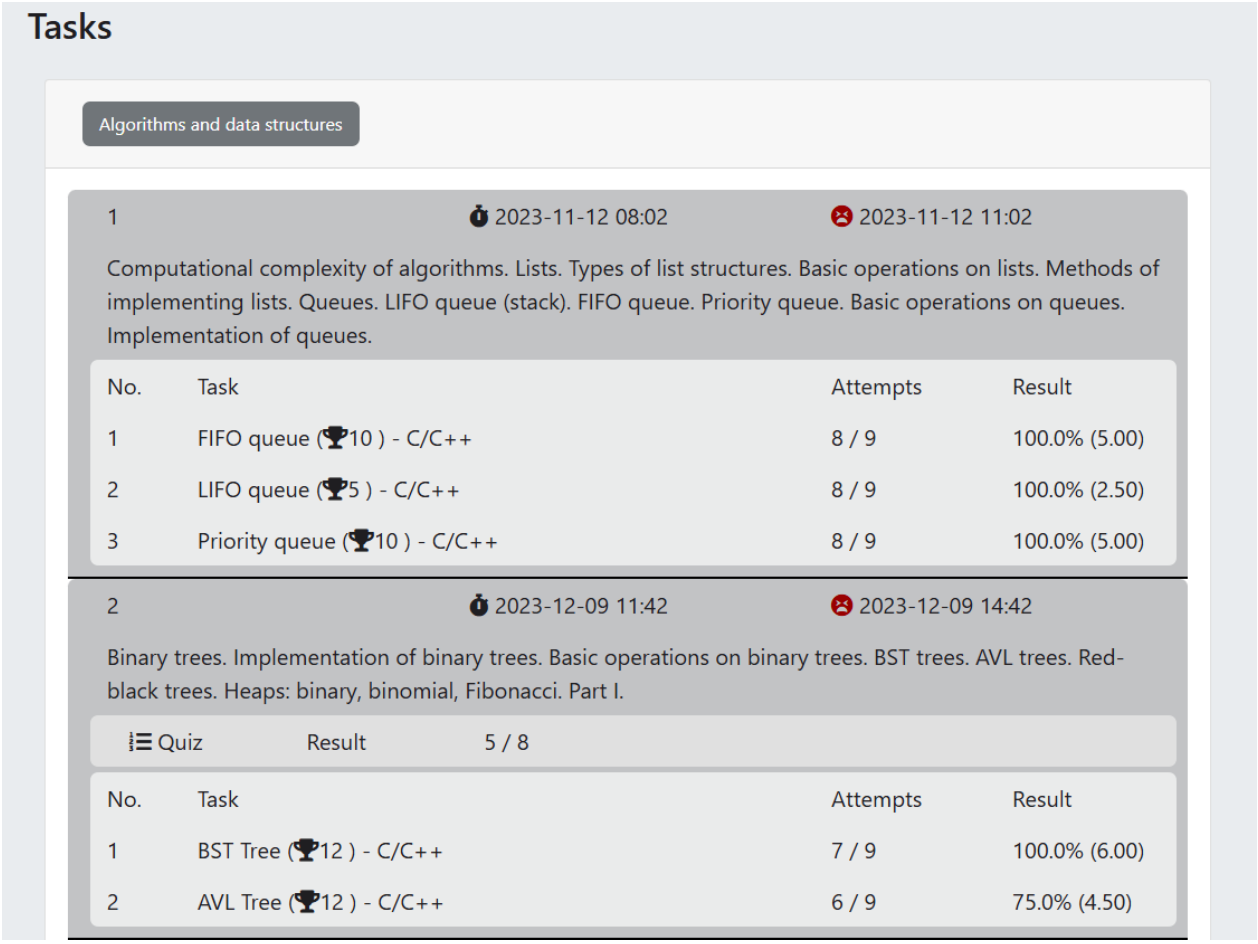

Fig 2. Student dashboard displaying lessons and associated programming tasks

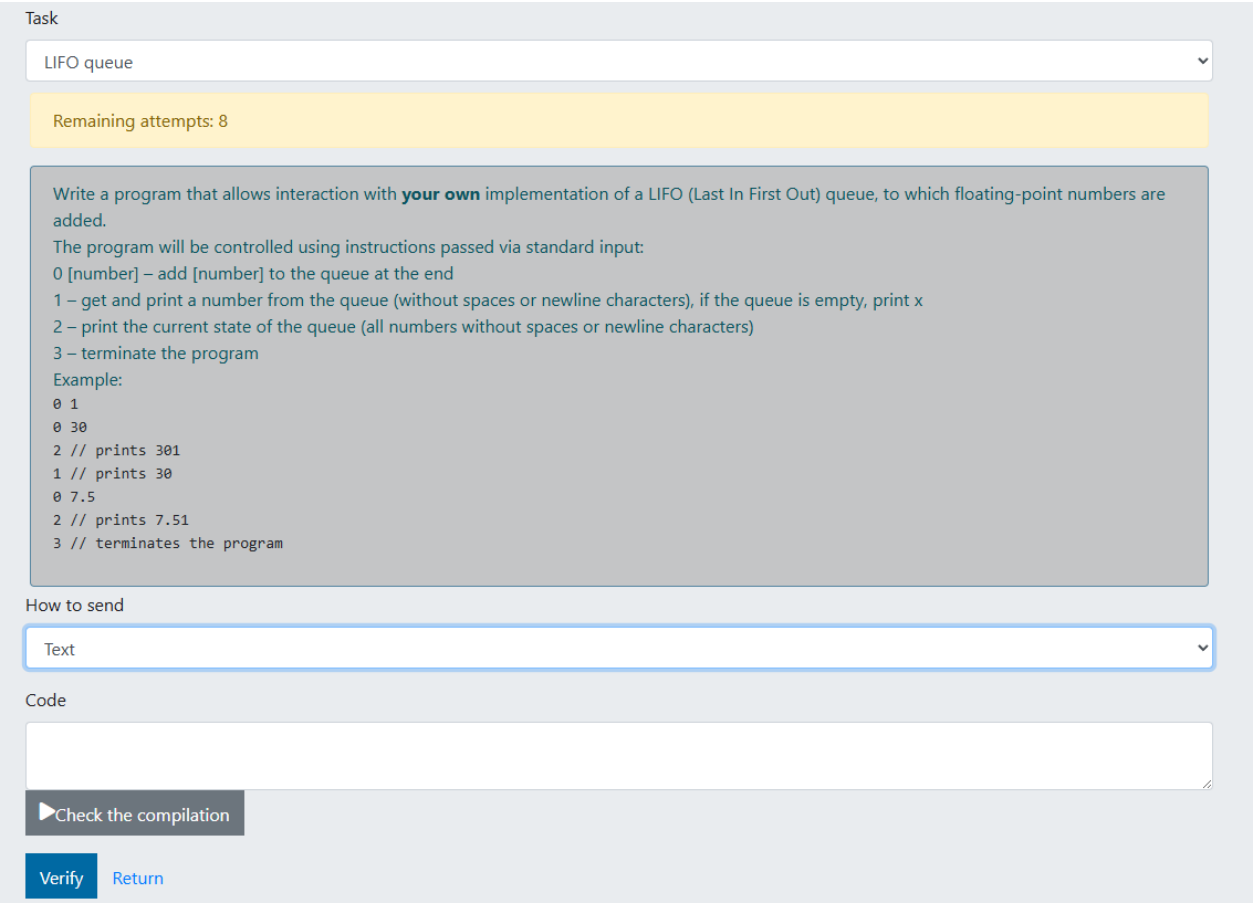

Fig 3. Assignment interface showing task description, code submission options, and submission status

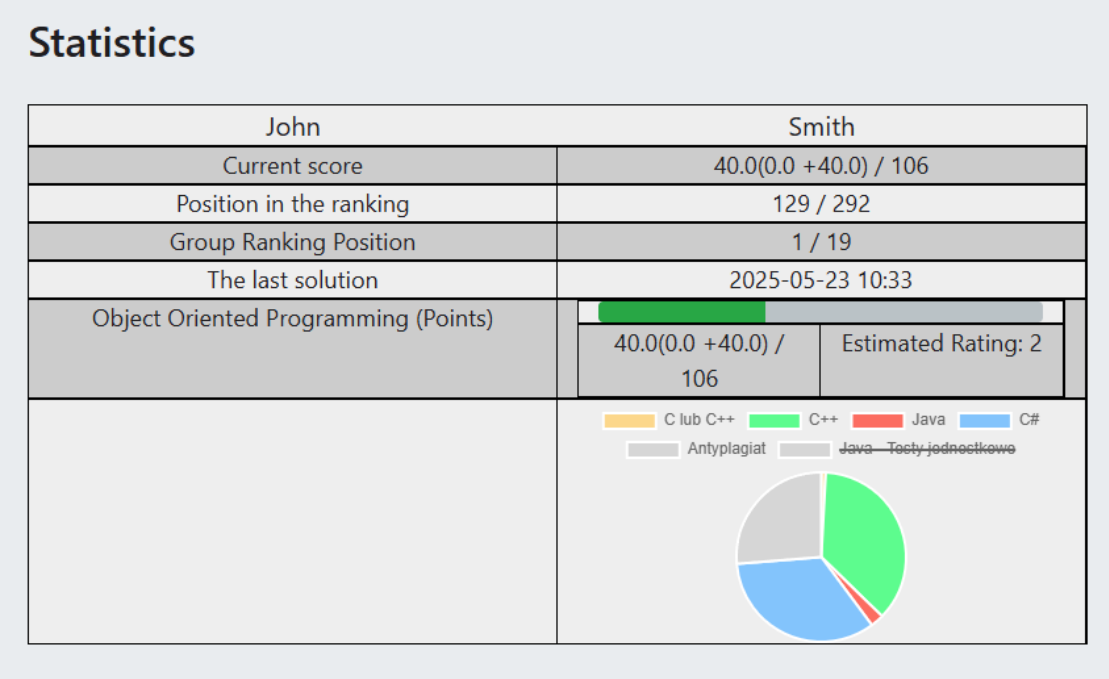

Fig 4. Student interface displaying assignment performance statistics, including scores and grades

Teacher functionalities:

- Define subjects, semesters, and student groups,
- Manage class realizations by combining subjects, semesters, and student groups,
- Create and organize lessons within each class realization (as presented in Fig 5),
- Define tasks (coding exercises) for students to complete,
- Automatically perform initial grading of submitted solutions,
- Review and manually grade submissions when needed,
- Compare student solutions to detect similarities (as presented in Fig 6),
- Assign final grades based on individual performance,
- Share teaching materials and additional resources with students.

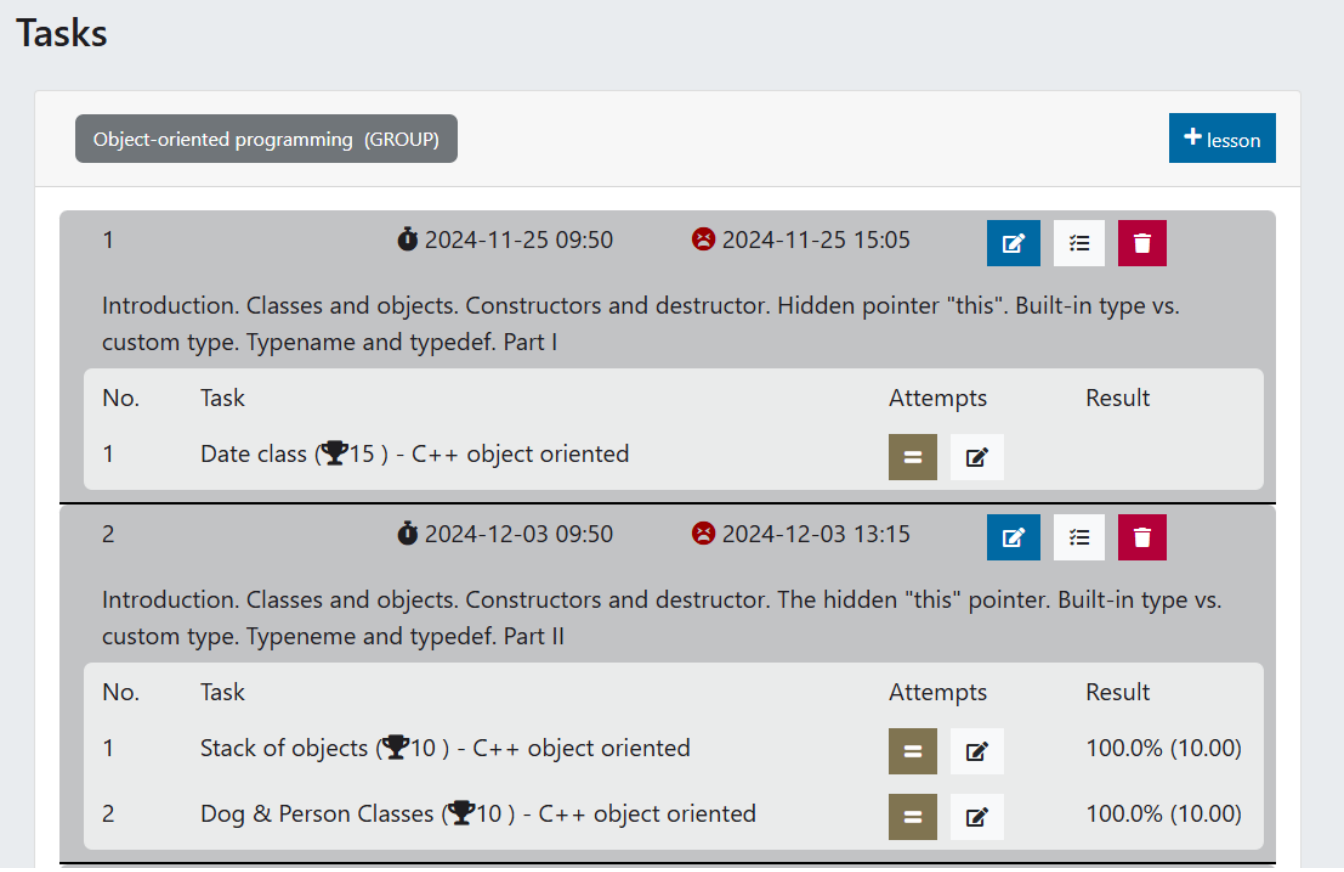

Fig 5. Teacher interface for managing class realizations, organizing lessons, and assigning tasks

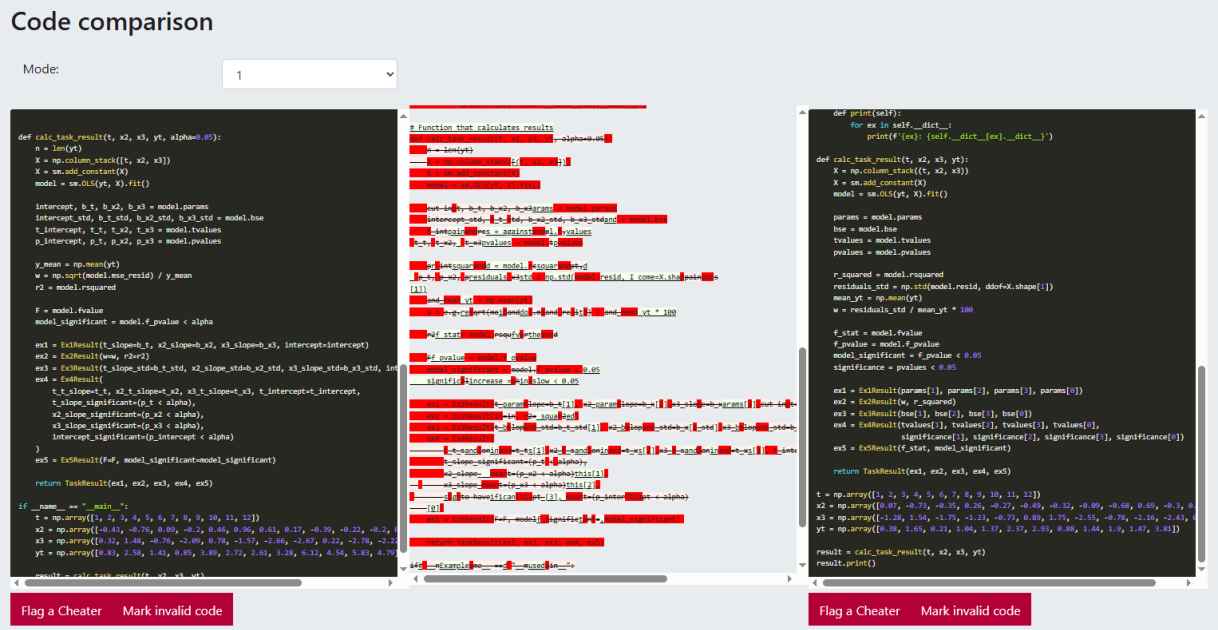

Fig 6. Plagiarism detection interface displaying code similarity comparison between student submissions

The most important functionality is the initial grading process of assignments, which is divided into eight steps.

1. File upload
   The student can upload a file – with an extension consistent with the selected programming language, a zip file with a project, or can directly paste the code into a text field. Once the solution is submitted, the system starts initial preparation.
   - For code provided by the text field a new file is created with the file format consistent with the programming language.
   - When a single file is provided, it is directly saved on the server.
   - When a zip archive is uploaded, it is unpacked in the target directory.
2. Preparation for plagiarism check
   A single file is created with all of the content extracted from text-based files (based on the file extensions).
3. Initial plagiarism detection
   The created file is compared with files generated within submissions of other students for the same assignment. For every pair a score is calculated representing the similarity between texts using Levenshtein distance (Pseudocode 1) (Su et al., 2008). Levenshtein distance was selected for its ability to capture minor differences in code structure while remaining computationally simple. However, it may not detect logic-preserving but syntactically altered code.
4. Online code similarity check
   Using http requests parts of the code (of specified length) are searched for online. In similar code sections are found, the similarity score is calculated again, but in this case against the code found online.
5. Code modification (optional)
   Depending on the programming language and task type, the code may be modified to allow verification. This may include:
   - Appending testing code,
   - Overriding the main method.
6. Environment creation
   To ensure system security and prevent code from affecting the host environment a docker container is started, this is where the code is compiled and executed.
7. Compilation (optional)
   If the programming language requires compilation, the code is compiled into a new file.
8. Execution and scoring
   The program is executed a predefined number of times with different inputs. The outputs provided by the program are compared to the expected results. This is used to calculate the final score. The types of verifications for different programming languages are placed in Table 1.

Pseudocode 1 The algorithm used to calculate the similarity score between submissions

```
function CalculatePercentageSimilarity(s1: String, s2: String): Float
    s1 ← toLowerCase(s1)
    s2 ← toLowerCase(s2)
    m ← length(s1)
    n ← length(s2)
    D[0..m][0..n] ← new integer matrix

    for i ← 0 to m do
        D[i][0] ← i
    end for

    for j ← 0 to n do
        D[0][j] ← j
    end for

    for i ← 1 to m do
        for j ← 1 to n do
            if s1[i−1] = s2[j−1] then
                D[i][j] ← D[i−1][j−1]
            else
                insertCost ← D[i][j−1] + 1
                deleteCost ← D[i−1][j] + 1
                substituteCost ← D[i−1][j−1] + 1
                D[i][j] ← min(insertCost, deleteCost, substituteCost)
            end if
        end for
    end for

    distance ← D[m][n]
    maxLength ← max(m, n)
    similarity ← abs(1.0 − (distance / maxLength)) × 100
    return round(similarity)
end function
```

Table 1: Verification methods available for different programming languages

Programming language	Verification method	Description
C/C++	Standard input and output	Pairs of input and expected output are provided by the teacher. The program is executed for every pair and the system checks if the output matches the expected value.
	Execution with parameters	Pairs of parameters and expected output are provided. The program is executed with different sets of parameters and the output is validated.
Java	Overriding the main method	The teacher provides the main method and it is added or used to replace the existing one. The method contains code that verifies if the submission is correct and can output the score value.
	Executing unit tests	The teacher defines unit tests and based on the number of passed tests the score is calculated.
C#	Overriding the Main method	Similar to Java.
	Executing unit tests	Similar to Java.
Python	Appending test code and reading standard output	Code provided by the teacher is appended to the submission and it should verify the correctness of the solution, it can output the score value.
	Executing unit tests	Similar to Java.

Database Design

The structure of the database is created using Hibernate ORM, which generates tables based on entity classes defined in Java. These tables are designed to store data about students, teachers, lessons, tasks and code submissions. Fig 7 presents the diagram for the tables and their relationships.

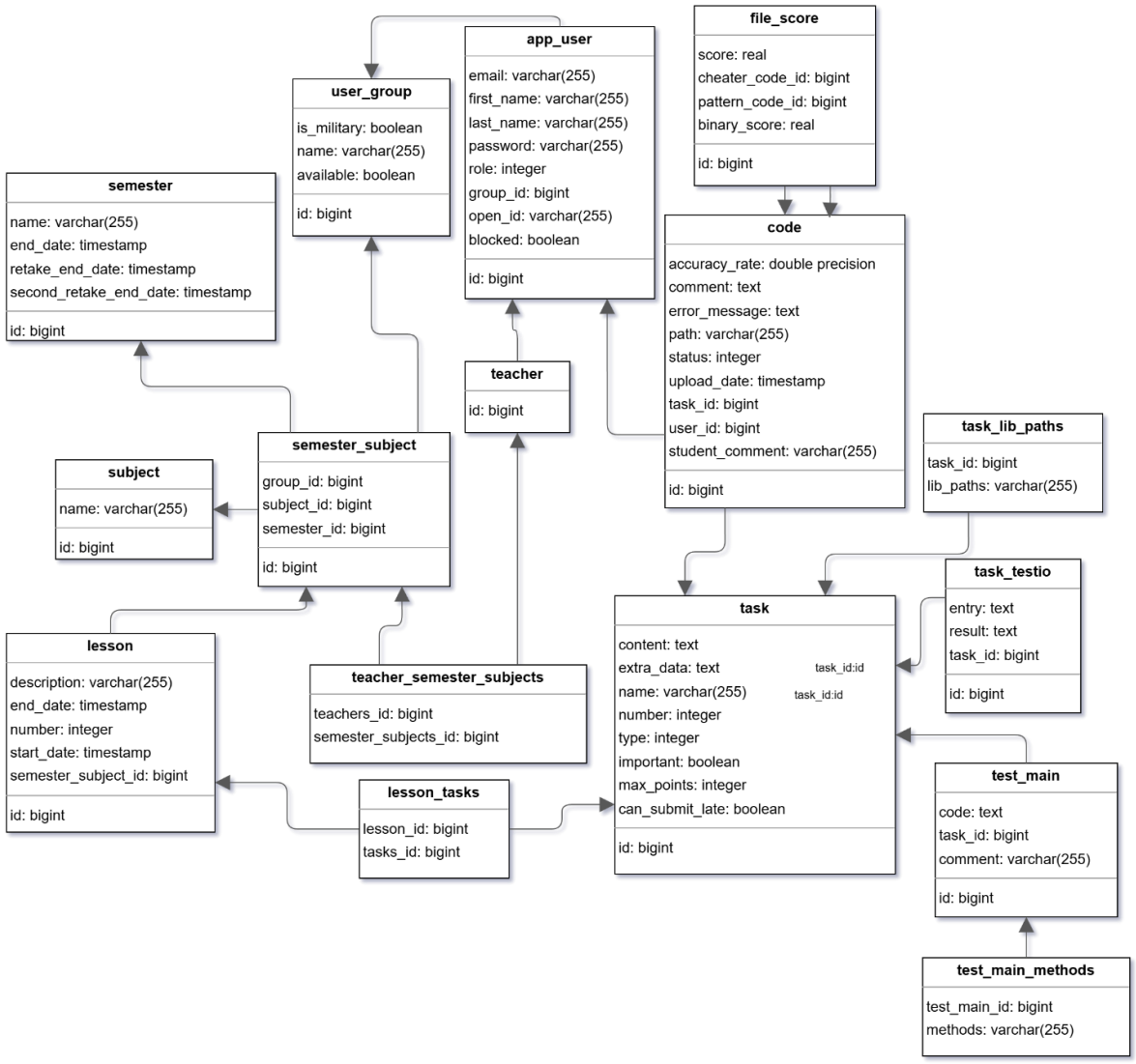

Fig 7. Entity-relationship diagram showing database structure for users, assignments, and submissions

The database centers around four main tables: *app_user*, *lesson*, *task* and *code*. All of the user accounts are stored in the *app_user* table, these can be students, that are assigned to a student group and teachers, which have access to a list of *semester_subject* rows – representing class realizations (consisting of a *semester*, *group* and a *subject*/course). Each *semester_subject* can have many lessons, with the date and time assigned based on the schedule. For every *lesson*, the teacher can assign multiple *tasks*/assignments, which the students are required to complete. The *tasks* are represented by the instructions, type and maximum number of points. Based on the selected verification method they can have assigned:

- additional libraries that are imported,
- pairs of inputs and expected outputs,
- main methods definitions,
- additional data required for other types of verification.

For every task, the student can send multiple submissions. Every attempt is inserted into the *code* table. Apart from information about the author, task and upload date it stores data regarding the verification, such as the status, accuracy result and error messages. Similarities between code submissions are stored in the *file_score* table.

Results

The developed platform has been actively used across six academic semesters over a period of three years, supporting the teaching of eight different courses, including:

- Algorithms and Data Structures,
- Object-Oriented Programming,
- Fundamentals of Simulation,
- Introduction to Programming,
- Methods and Techniques of Computer Simulation,
- Programming Languages and Techniques,
- Internet of Things Technologies,
- Forecasting Methods.

During this time, over 300 students submitted a total of 2,856 programming assignments across 61 unique tasks. The system has significantly contributed to the educational process by making the processes of submission, evaluation, and grading faster and more efficient.

Students particularly appreciated the automatic grading feature, which provides instant feedback after each submission, allowing them to make improvements in their solution until the deadline. Many students also found the system's hints useful in identifying mistakes and improving their code, which allowed them to achieve better results and helped them in the studying process. For the teachers, the platform has proven to be an effective and time-saving solution. Automatic verification allowed to check if the implemented algorithm works as expected, in all of the defined cases. Plagiarism check made it possible to find similar solutions and detect dishonest work, which would be very difficult and time consuming especially when the problem would occur between students of different groups. This allows the teacher to focus more on teaching quality and individual support.

The system has remained stable without any significant downtimes, and the students have not reported any issues with access to the application, probably due to the limited but manageable scale of usage. Assignments are typically auto-graded within a few seconds, depending on their complexity and verification type. During the life time of the system the students reported some bugs, mostly related to the definition of specific assignments rather than the system, as well as suggested some ideas to improve the experience. This has led to new features and updates carried out since its initial deployment of the platform. New programming languages were added to expand compatibility and usability for different courses, plagiarism detection mechanisms were enhanced, and manual evaluation options were introduced to complement automatic scoring. Additional features like inline hints, feedback, and the ability to share educational materials have also been implemented, making the platform more interactive and helpful.

Discussion and conclusions

The goal of this project was to create a web platform that simplifies and automates the evaluation of programming assignments, particularly in academic settings. The system combines three key functionalities: the creation of programming tasks, automated grading based on predefined test cases, and detection of code similarity between student submissions. These elements are integrated within a single platform that does not require external tools or plugins, which sets it apart from many existing solutions used in academic environments.

The implementation and deployment of the system during university classes has confirmed its usability. It was possible to prepare tasks and define verification without spending a lot of time, and the automatic grading feature significantly reduced the time needed to evaluate submissions. The grading process was fair and reproducible, with detailed feedback available for each student, improving the transparency and consistency of the evaluation process. The interface also proved intuitive, both for teachers and students.

An important feature of the system is its built-in similarity detection, which helps identify potential cases of plagiarism. Although the approach is relatively simple, it proved effective in detecting suspiciously similar solutions, especially when used as a first-level filter for simple coding assignments. This functionality is particularly valuable in large groups, where manual inspection of all code submissions would be impractical. It contributes to promoting academic honesty while respecting the privacy and autonomy of students.

Despite its success, the system has some limitations. The current similarity detection could benefit from more advanced techniques, such as abstract syntax tree comparisons or machine learning-based approaches, to better handle cases of superficial code obfuscation. Additionally, support is currently limited to a subset of programming languages, which restricts its use in courses that require less common languages or multiple paradigms. Addressing these limitations would further increase the platform's versatility and robustness.

In summary, the developed platform has proven to be an effective and efficient tool for managing programming assignments in academic context. By reducing teacher workload, providing timely feedback to students, and supporting basic plagiarism detection, it improves both the teaching and learning experience. While there is room for improvement, especially in terms of similarity analysis and language support, the current version already delivers substantial value and provides a strong foundation for future development.

References

- Agrawal, A., Jain, A., & Reed, B. (2022). CodEval: Improving Student Success In Programming Assignments. EDULEARN22 Proceedings, 1, 7546–7554. https://doi.org/10.21125/edulearn.2022.1767
- Bowyer, K. W., & Hall, L. O. (1999). Experience using "MOSS" to detect cheating on programming assignments. Proceedings - Frontiers in Education Conference, 3. https://doi.org/10.1109/FIE.1999.840376
- Codio | The Hands-On Platform for Computing & Tech Skills Education. (n.d.). Retrieved May 29, 2025, from https://www.codio.com/
- Dobre, I. (2015). Learning Management Systems for Higher Education - An Overview of Available Options for Higher Education Organizations. Procedia - Social and Behavioral Sciences, 180, 313–320. https://doi.org/10.1016/J.SBSPRO.2015.02.122
- GitHub - thanhtri/plagiarism_programming: A plugin that integrate MOSS and JPlag to Moodle and provide other useful features. (n.d.). Retrieved May 29, 2025, from https://github.com/thanhtri/plagiarism_programming
- Lalani, K., Crawford, J., & Butler-Henderson, K. (2025). Academic leadership during COVID-19 in higher education: technology adoption and adaptation for online learning during a pandemic. International Journal of Leadership in Education. https://doi.org/10.1080/13603124.2021.1988716
- Lobb, R., & Harlow, J. (2016). Coderunner. ACM Inroads, 7(1), 47–51. https://doi.org/10.1145/2810041
- Oliveira, G., Grenha Teixeira, J., Torres, A., & Morais, C. (2021). An exploratory study on the emergency remote education experience of higher education students and teachers during the COVID-19 pandemic. British Journal of Educational Technology, 52(4), 1357–1376. https://doi.org/10.1111/BJET.13112;PAGEGROUP:STRING:PUBLICATION
- Randall, J., & Engelhard, G. (2010). Examining the grading practices of teachers. Teaching and Teacher Education, 26(7), 1372–1380. https://doi.org/10.1016/J.TATE.2010.03.008
- Sağlam, T., Hahner, S., Schmid, L., & Burger, E. (2024). Obfuscation-Resilient Software Plagiarism Detection with JPlag. Proceedings - International Conference on Software Engineering, 264–265. https://doi.org/10.1145/3639478.3643074;PAGE:STRING:ARTICLE/CHAPTER
- Su, Z., Ahn, B. R., Eom, K. Y., Kang, M. K., Kim, J. P., & Kim, M. K. (2008). Plagiarism detection using the Levenshtein distance and Smith-Waterman algorithm. 3rd International Conference on Innovative Computing Information and Control, ICICIC'08. https://doi.org/10.1109/ICICIC.2008.422