Duarte André Teresa Guerreiro

CodeInsights
Final Report

1290 UNIVERSIDADE D COIMBRA

UNIVERSIDADE D
COIMBRA

Duarte André Teresa Guerreiro

CodeInsights

Final Report

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Professor António José Mendes to Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

CodeInsights

Final Report

Duarte André Teresa Guerreiro

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. António José Mendes and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering..

October 2021

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Abstract

Due to the world's growing dependence on technology, Computer Science has become a highly sought after path for new college students. These types of courses present an alarming high drop rate, and for a long time the origin of the students' success/failure has been investigated. The teachers' close supervision has been identified as a significant element in students' success. However, as the number of students increases, classrooms get increasingly larger, making it more difficult for teachers to keep track of the entire class's progress and issues.

Programming, which is the foundation for Computer Science, has also been identified as a challenge for students. Students can quickly learn the basic building blocks of programming, but often struggle when asked to arrange them together in the correct way to solve an assignment. This problem-solving skill is acquired by extensive practice, which can take years. Time some students might not have, and consequently drop out of the course.

CodeInsights is one of several tools that have been developed to help with the difficulties of teaching and learning programming. This real-time monitoring tool is capable of capturing real-time copies of the students code. The snapshots are processed, and a series of visualizations and aggregated data is made immediately available for teachers.

Like any other piece of software, CodeInsights is in an neverstoping cycle of improvements. Many instructors who use the system have provided a wide array of suggestions for a future version of this tool. So the aim of this thesis was to improve CodeInsights based on this feedback, by providing a set of features to better facilitate the teaching process. The major features analysed, designed and implemented were: 1) A chatting system to facilitate the communication between teachers and their pupils 2) A sharing code component for teachers to better address students error 3) A security layer to block the access of the student's code. In the end, all the main objectives of this dissertation were achieved and soon the improved version of the system will be available to the new users of CodeInsights.

Keywords

Programming Education, Monitoring Tool, Visualization.

This page is intentionally left blank.

Resumo

Devido à nossa grande dependência com a tecnologia, Engenharia Informática é cada vez mais um curso com grande procura. Este tipo de cursos tem vindo a apresentar um alarmante número de desistências e por isso várias investigações sobre a sua origem foram realizadas. A forte ligação entre professor e estudante é um fator determinante no sucesso dos estudantes. Porém visto que o número de alunos em cada sala de aula é cada vez maior, o trabalho de supervisionar os alunos torna-se mais difícil.

A programação encontra-se também na origem das várias dificuldades dos alunos. Os alunos conseguem facilmente aprender as estruturas básicas de programação, mas apresentam uma maior dificuldade ao juntar essas peças para resolver os exercícios/problemas de programação. Esta habilidade de resolver problemas leva anos a ser masterizada, tempo que alguns alunos podem não ter.

O CodeInsights é uma das vastas ferramentas que foi desenvolvida para apoiar os professores a monitorizar os seus alunos. Esta ferramenta é capaz de capturar cópias do código dos alunos em tempo real, e produzir imediatamente uma série de gráficos que ajudam o professor a identificar problemas entre os seus alunos.

Como a maioria do *software*, existem sempre melhorias a serem realizadas e novas funcionalidades a serem implementadas. Foi este então o objetivo desta dissertação, melhorar o sistema com base numa série de sugestões feitas pelos utilizadores do CodeInsights. As principais funcionalidades analisadas, concebidas e implementadas foram: 1) Um sistema de *chat* para facilitar a comunicação entre os professores e os seus alunos 2) Um mecanismo de *code sharing* para que os professores possam demonstrar aos seus alunos maneira de resolver exercícios 3) Uma camada de segurança para bloquear o acesso que código dos alunos tem sobre o sistema. No geral todos os principais objetivos deste projeto foram conseguidos e em breve a versão melhorada do sistema será disponibilizada aos novos utilizadores CodeInsights.

Palavras-Chave

Educação em Programação, Ferramenta de Monitorização, Visualização.

.

This page is intentionally left blank.

Acknowledgements

I would like to use this section to thank everyone who was involved and helped me during this project. I am sure the lessons I learned throughout this project will follow me for the rest of my career. In no particular order I would like to thank the following people:

- António Mendes: For being my tutor and providing crucial feedback based on his vast experience. Thank you for your guidance throughout this project.
- Nuno Gil: Nuno was my supervisor away from António, and helped me through all the stages of the project. Thank you for your availability with the CodeInights matters.
- FCT: Foundation of Science for technology, I.P., within the scope of the project CISUC - UIDB/00326/2020, for financially supporting this project.
- I would also like to thank all my family who showed me emotional support every step of the way until this very moment. Here I would like to thank my mom in particular, as she is very vocal about these things and I could not get away with not doing so. Thank you for being what every mother aspires to be.
- Additionally thanks to all my friends who keep up daily with my humoring and all the people who gave me strengthening words to keep going.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Glossary

AJAX Asynchronous JavaScript and XML. 33, 34

API Application Programming Interface. 6, 36, 47

AST Abstract Syntax Tree. 31, 46

CRUD Create, read, update and delete. 23

CS Computer Science. 1

DOM Document Object Model. 37, 38

GUI Graphical user interface. 6, 39, 40

HTTP Hypertext Transfer Protocol. 33, 34, 35, 47

IDE Integrated development environment. xiv, 5, 6, 8, 12, 13, 17, 18, 19, 20

LMS Learning management system. 5, 20

OWASP Open Web Application Security Project. 28, 29, 30

URL Uniform Resource Locators. 13, 29

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

1.1 Background

With the great improvements to technology and information, the environment around us has changed. Now more than ever, technologies are a part of our world and our everyday lives, whether in the form of smartphones, computers, or any other kind of device. Like Lukas et al. [42] states *'a future where many things will be connected to the internet seems increasingly palpable'*. Due to the world's large dependency on technology, having programming skills has become a very useful ability and therefore in high demand. According to a survey [2] conducted in 2013 software development, one of the many branches of Computer Science, (CS) was among some of the highest-rated jobs. Unsurprisingly year after year, CS maintains its status as one of the most attractive areas for new coming college students.

Research [27] shows that during this transaction period, introductory courses are critical for a student's success, as they form the entire course's groundwork. Normally a course is structured in a hierarchical manner, and students progress through it. In the earlier parts of the course, students learn lower level skills and will eventually progress upwards. Difficulties in these premature phases propagate to the levels and subjects above, leading to many students feeling helpless and in some cases giving up on their studies. For the specific case of CS, programming takes a vital role in this course's educational program, so it is really common to see these two terms being related as they go hand in hand.

1.2 Context and Motivation

As CS courses became increasingly more popular, some efforts were made to discover the nature of students' difficulties in the area of programming. These difficulties can range from factors such as mathematics skills, parents' education to simply lack of motivation. We must emphasize the nature of programming as one of the sources for the students' challenges.

In some ways, learning how to program could be seen as learning a new language, the language of computers [7]. However, the programming process cannot be decomposed into simple syntax learning, as a major barrier to students is the prior thought process needed to solve a certain problem [3]. Programming is not about writing code in a certain language, whether that would be Java, Python or any other, but it is rather about the process of understanding a problem and creating an algorithm that fits that type of problem.

Therefore, programming languages must be seen as means to an end and not an obstacle. Students develop their problem-solving skills as they progress through learning and beyond, something that could take years since programming is a highly repetitive and practical exercise.

Another factor that plays a major role in the student's ability to learn, is their working habits, or as according to Norris et al. [35] the lack of them. In their survey, it is identified a critical disparity between students' and instructors' perceptions. For example, instructors universally complain about the student's lack of testing, but generally, students agree that they regularly test their code. Arto Vihavainen et al. [50] noted a different perspective but a similar conclusion, attributing student's success to early momentum, built up from self-confidence and a sense of routine.

In essence, learning the building blocks of programming and using them in a meaningful way to achieve a correct solution is not an easy task. However, neither is teaching, as instructors must cope with many students with very different personalities and behaviors. To make matters worse, some of the students who face these issues might not even try to reach out for the instructor's help, making them totally unaware of such difficulties. For instance, some students avoid exposing their questions and problems because they are concerned that their peers will interpret it as a weakness [14].

It is easy to see that there are a lot of factors in the whole learning environment, which can easily add up, making this process one-off high complexity. This is the base problem that many monitoring tools over the last few decades have been addressing. This dissertation instead of pursuing a more usual approach of developing a system from scratch will build upon an already functional system called CodeInsights. The next section we better explain the goals of the new improved version of this system.

1.3 Goals

CodeInsights is a tool developed for addressing the complexity problem inherent to teaching programming, its purpose is to provide critical support to instructors in identifying arising problems among students. This system is a web-based tool capable of monitoring students, by collecting snapshots (i.e., essentially copies) of their code no matter where they are working. By collecting and storing a large number of snapshots the tool is capable of providing detailed information and visualizations to the teachers about each of their student's programming processes.

CodeInsights was already used several times in the past and has been gradually improved with the feedback provided by instructors and students who used it. At the beginning of this project, the client/owner of the system had a list of such observations that served as a foundation for the new version of the system. Thus the goals of this dissertation were as follows:

- Learn about the existing tool and the underlying monitoring tool environment.
- Analyze, plan and implement improvements and new functionalities for the enhanced version of the system.
- Carry out tests on the improved tool and enhance its security.

1.4 Structure

This dissertation is organized in the following order:

- **Chapter 2 – CodeInsights:** This chapter looks at the previous state of CodeInsights and details its major functionalities and components.
- **Chapter 3 – Related Work:** The next chapter goes over the programming monitoring tools ecosystem, while comparing some of its most popular tools with CodeInsights.
- **Chapter 4 – Requirements:** The following chapter determines the system's enhancements using a requirement engineering approach. In here the functional requirements, quality attributes and constraints are defined.
- **Chapter 5 – Design:** This chapter looks at some design mock-ups as well as a high level architecture of the improved version of the system.
- **Chapter 6 – Implementation:** This chapter shows how the new functionalities were built, and how some problems and barriers that arose were dealt with.
- **Chapter 7 – Planning:** This chapter summarises the project development plan and shows its expected vs actual timeline.
- **Chapter 8 – Conclusion:** Finally, the last chapter summarises the whole project, while taking a retrospective view of the realized work.

This page is intentionally left blank.

Chapter 2

CodeInsights

The former state of the CodeInsights system is described in this chapter. More specifically we start in section 2.1 with a brief background on how programming courses are structured and taught resulting in lost information. Next in section 2.2, we explain the basic IDE-plugin approach used by the system to capture the student's source code. With the basic functionality of the system described we move to the presentation of the CodeInsights architecture and all its components in section 2.3. Finally, under section 2.4, all the remaining system capabilities, such as plagiarism and notification, are described.

2.1 Background

As mentioned in the previous chapter, programming has a hierarchical structure where teachers will start by introducing easier concepts to their students and will gradually move towards more complex ones. This sense of progression is reflected in programming worksheets, which are fundamentally a set of assignments to be completed by the students. For example, to keep students engaged, the early activities on a worksheet should be simple and the subsequent ones more difficult. Worksheets are organized in a numerical faction, where each task is given a certain number (e.g., 3). Additionally, each task can be divided into many sub-tasks to aid the student's understanding of the problem. For instance, 3.1 and 3.2, and so forth.

In order to aid the teaching process a lot of courses make use of a type of tool called LMS. An LMS [15] or Learning Management System is a software application that automates the administration, tracking, and reporting of training events. There are several types of LMSs, some open-source and others not. Nevertheless, a few of the most popular are: Moodle, TotalLMS, Blackboard Academic Suite, and Plateau LMS. Typically these systems are responsible for storing and displaying programming worksheets.

In short, the student's workflow is as follows: they download some sort of worksheet from an LMS. After having the description of each problem, they start solving it in their favorite or recommended IDE (Integrated development environment). Their work typically stops when they find a solution for the assignment or simply get stuck. Thus no submission is ever made, meaning that instructors only have references to the student's work in more formal evaluation, like projects and exams. Only having information about the student's challenges so late means that there is little to nothing to be made.

2.2 CodeInsights Aproach

Rather than losing a large amount of information by discarding a lot of the student's code and its multiple iterations, *what can be done to capture it?* This was the premise of CodeInsights, trying to capture as much of the student's code as possible whenever and wherever. Students use a special type of software to write their code, the so-called IDEs. An IDE [22] or integrated development environment is a software for building applications that combines common developer tools into a single graphical user interface (GUI). IDEs [10] combine all the common activities required to write software into a single application: editing source code, building executables, and debugging. Fortunately, the functionality of these systems may be expanded through the use of plugins. Plugins can be pre-built and there is a vast number of them available on the internet. Alternatively, they can be created from scratch and tailored to the user's needs through the use of an application programming interface (API).

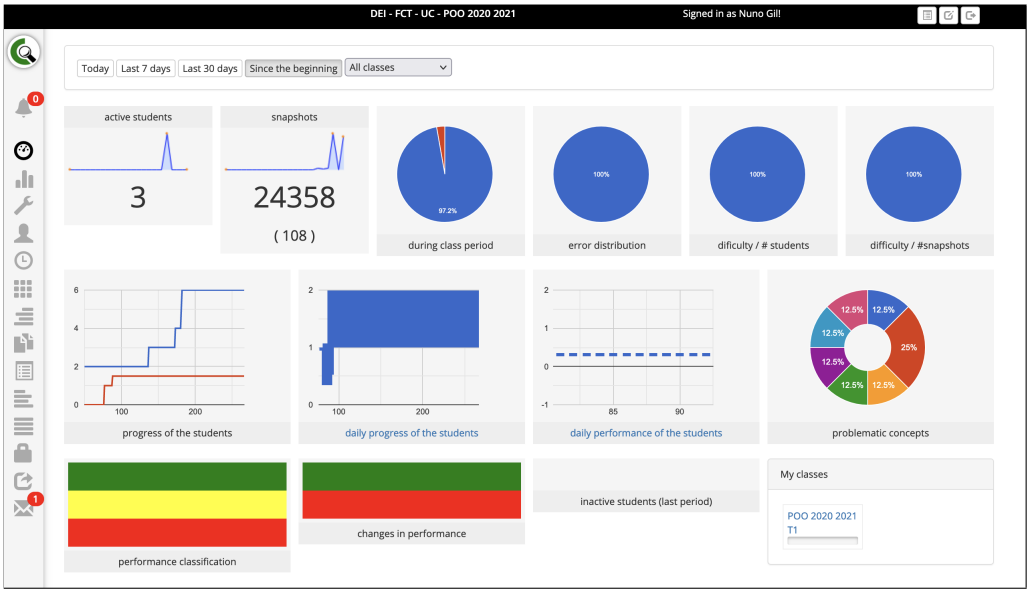


Figure 2.1: Main Dashboard of CodeInsights

By using custom plugins CodeInsights can record snapshots of the student's code every time a major event occurs [16]. This is extremely important as it provides the system with a real-time quality, meaning teachers have almost instantaneous access to the student's code. Generally speaking, as it may vary depending on the IDE, a snapshot is captured every time the code is compiled, saved or ran. Even though a snapshot is made up of text that represents computer commands, it contains critical information for evaluating a student's performance. For instance, a snapshot is composed of the number of code lines, the validity of the code, and even the date when it was submitted. Code validity is particularly important for the teachers, and CodeInsights checks it by running the student's code against a set of test cases.

With this valuable information, the system can create a visualization to support the instructor's analysis and notify the instructor of the occurrence of desired events. In figure 2.1 the main dashboard of the system is partially represented. To make this page as simple to grasp as possible, it offers an aggregated view of the system's most essential captured metrics. For instance, one of the many visualizations is a pie chart dedicated to repre-

senting the number of snapshots captured during a class (the schedule of the classes is a parameter used in the system and managed by the teacher).

2.3 System Architecture

Now that the basic operation of the system is explained, we can now describe the previous CodeInsight's architecture with all its components. CodeInsights can be broken down into two main components as observed in 2.2, the client and the server-side. The server side is responsible for all the processing and logic involved in bringing information to help instructors make educated guesses on which students are facing problems. Furthermore, the server-side can be decomposed into several components: snapshot processing, sandbox server, database, notification system, monitoring tool, and plagiarism component.

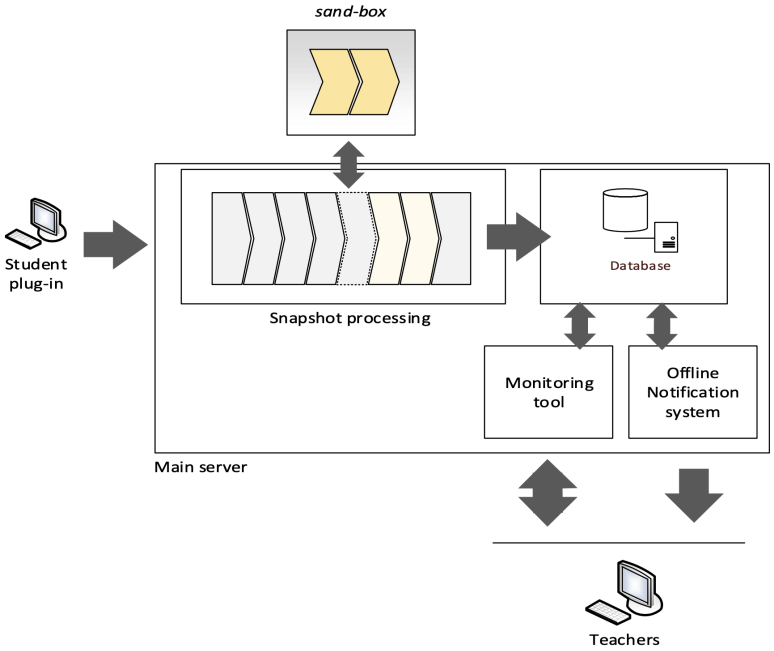


Figure 2.2: Previous architecture of CodeInsights

The snapshot processing component is in charge of taking student snapshots as inputs and sending their data (i.e., the student's source code) to the sandbox server. The sandbox server is the system component in place to determine whether or not the source code is valid. This component evaluates the code against a set of test cases that have already been defined. The database is the system's heart, and it is responsible for storing the code validation process' outputs. The monitoring tool component, which uses all of the data saved in the database, provides a set of visualization and other features to aid the teaching process. The system's notification component duty is to notify the teacher when certain important events occur.

As for the client-side, its responsibility is to let users interact with the system and benefit from its functionalities. It is essentially a graphical user interface built with JavaScript and HTML. When inspecting the client-side, it can be further decomposed into two views, the instructor's and the student's.

Before advancing to the next section it must be pointed out that the system closely resembles a two tier architecture [23]. This type of architecture is defined by the separation of data presentation and data business logic and management. In the first tier resides the client interface layer. The database server and web application server reside on the same server machine, which is the second tier.

Theoretically, to provide optimal security the sandbox should run on its independent server, as seen in figure 2.2. Since the sandbox server runs code from strangers, by being in a separate machine even if some malicious code is run, the damage is mitigated to that host only. However, this project only has access to one machine, meaning in practice the sandbox server and the main web server together with all of its components, run on the same host.

2.4 System Features

Based on the data that is collected about students' programming activities, CodeInsights provides numerous functionalities for both the instructors and the students. In section 2.2 we defined a high-level overview of the system and explained the fundamentals of CodeInsights. However, this is a simplistic approach in representing the system, considering its complexity and all its functionalities. Therefore in the next set of subsections, we explain in more detail each one of CodeInsight's features.

2.4.1 Data Collection

Students have two ways of submitting their code, they can either use the online text editor located on the student's view in the web application. Or by installing a plugin in their favorite IDE, which automatically submits their code whenever they run or save a program. Currently, the system supports Atom, Eclipse and JetBrains IDEs (such as PHPStorm, IntelliJ or PyCharm). Because of the 'silent' manner in which these plugins operate, this is the recommended method since it has no impact on the student's programming.

Before the students' code starts being utilized by the system some tweaks to the source code are required. The required changes will be discussed in the next parts of this section.

Annotations
A snapshot is associated with two other entities in the system: the student who submitted it and the assignments for which it was coded. As it stands the system only receives source code and has no method of linking a snapshot to an assignment and a student. CodeInsights uses a really simple, yet highly reliable way of linking these three elements by using code annotations. An annotation is nothing more than extra information associated with a piece of code. It can be a note that includes a comment or explanation.

CodeInsights will look for comments that start with the special character @ at the beginning of the source code. For example, if a student with id 9999 wants to submit code for assignment 3.1, he would need to add two comments at the beginning of their code with the following annotations: @sid 9999 and @aid 1.1. Because students may forget or misspell annotations, the system will continue to process the snapshot as an anonymous entity if there are no proper annotations.

Input comments

Associating snapshots to other entities is not the only type of source code alteration needed to use CodeInsights. Going back, and looking at the method of creating assignments, one of its many parameters are test cases. As explained before, test cases serve the purpose of validating the code (i.e., classifying it as correct or not). A test case is nothing more than a set of inputs/outputs that dictate how a certain program should work. By knowing the behavior of a program teachers can specify up to 4 test cases in CodeInsights. A 'simple calculator' program is a classical example, it receives two numbers and outputs their sum and multiplication. Table 2.1 exemplifies one of the many possible test cases for this simple program.

	Input	**Ouput**
Test case #1	int numero1 = 8; int numero2 = 3;	8 + 3 = 11 8 * 3 = 24
Test case #2	int numero2 = 8; int numero2 = 4;	8 + 4 = 12 8 * 4 = 32

Table 2.1: Test case examples

At the beginning of programming, a lot of concepts are still to be taught, some of these concepts are functions and their arguments. Functions [38] are 'self-contained' modules of code that accomplish a specific task. Functions usually 'take in' data, process it, and finally 'return'. Arguments are a way of providing information to functions in the form of variables. Since students do not have access to functions all their code is run on the main function. The main function serves as the starting point for program execution. By running code in the main function and without having the knowledge of its arguments it becomes very hard to make a simple input/output mapping.

CodeInsights utilizes a clever approach to avoid this problem by delimiting a specific zone where the inputs will be defined. The input zone is an area located at the beginning of the code where input variables must be defined. It starts with a *begin_inputs* comment and ends with an *ends_inputs comment*. Every time the system receives a snapshot, immediately after looking at the identifiers annotations, the system searches for the input zone. Once it is found, the system can then switch this area and its variables with the variables previously defined by the teacher while creating the assignment. At its core, this approach is conceptually really close to the notion of functions, but without its standard syntactic definition. One significant drawback from this mechanism is that teachers and students must use the same variable names, otherwise, the system will get 'use of undefined variable errors'. Since teachers utilize worksheets they can specify in each assignment the correct variable names to be used.

Figure 2.3 represents all the changes needed for the 'simple calculator' example to make it correctly processed by CodeInsights. Highlighted in blue are the annotations necessary for identifying the snapshot. Highlighted in red is the input area where the input variables will be switched for validation purposes. Table 2.1 indicates some of the test cases that could be defined by the teacher to validate this example. It is important to note that there is a match in variable names (numero_1 and numero_2) between source code and test cases.

Figure 2.3: Source code alterations example

2.4.2 Snapshot processing

The snapshot processing component is responsible for validating and collecting submission metrics. Upon receiving a snapshot, this component collects some metrics like the number of lines of code. Then depending on the language of the code, the server can either compile or interpret it. On the previous build of CodeInsights languages such as Java, JavaScript, C, Python were supported. If the compilation or interpretation presents errors, they are noted and stored for further display to the instructor. If not, the code can then be tested using the pairs of inputs/outputs defined by the instructor. The results of the test cases are categorized by the following rules:

- **correct :** If all the pairs of inputs/outputs match each other
- **incorrect :** If there is a mismatch between outputs
- **potentially correct :** If the outputs are not exactly equal to the expected values, but they are very similar

An important factor to the snapshot processing component is the code running time. It is usual, especially in introductory programming courses, for students to make overly complicated solutions or even mistakenly create infinite loops. Therefore if the code takes more than a certain number of seconds to run, it is classified as 'incorrect'.

2.4.3 Instructor View

Instructors can access the collected data through the CodeInsights instructor view which is available via a web browser interface. Here they have a diverse array of visualizations to help them understand their student's difficulties. Like it was discussed before, the main dashboard functions as the heart of the system. However, due to its aggregation property, some information is not directly shown in it. The missing visualizations are accessed in their dedicated areas, like the live submission feed graph. In addition, many of the system

visualizations and their data are linked, allowing for drill-down operations with a single button press. Drilling down is the ability to present the user with general data and allow him to filter it into more specific data.

For demonstration purposes, the next areas will explain in more detail some of the system's visualizations, as it is not feasible to detail every single one of them.

Snapshot Details

The page of CodeInsights where teachers can explore and see detailed information about each snapshot submitted is shown in Figure 2.4. To begin, the teacher must identify which student and which assignment he wishes to examine. Following that, she/he is presented with a number of details regarding the assignment submissions, such as the number of lines of code, its execution time and its output.

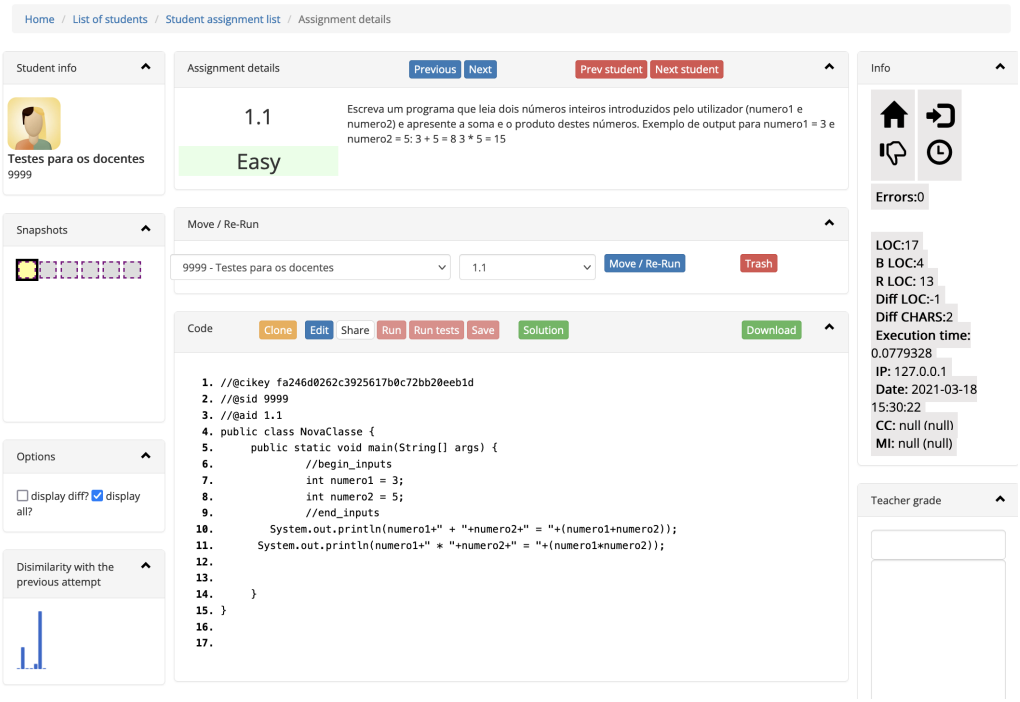


Figure 2.4: Teacher's view - Snapshot details page

One of the most important parts of this page is a timeline of labeled snapshots located on its left side. Each snapshot present in the timeline is color-coded to make the process of identifying incorrect source code easier. All the possible mappings between snapshot classifications and their color are as follows:

- **Grey:** Unidentified error.
- **Green:** The code has passed all of its test cases with success.
- **Yellow:** The code didn't pass all tests and only managed to succeed in some of them.
- **Orange:** The code failed in all of the assignment test cases.
- **Red:** The server could not run the code as it presented compilations errors.

- **Dashed Outline:** the code was submitted after the deadline that was defined for this assignment.

The snapshots are sorted by submission date, with the most recent one being chosen by default. The teacher can freely and seamlessly switch between snapshots with the click of a mouse on the desired submission. Additionally, a lot of other functionalities are available on this page. To name a few, teachers can: re-run student's code, edit student's code, classify student's code and send feedback to students.

Live Feed

If a teacher wants to see their students' work in real time, they can access the live feed section depicted in figure 2.5. Each student is represented by a line, and their last submission in n seconds is shown as a rectangle. Once again the submissions are color coded, and for consistency, they follow the same mapping as explained before. This feature is mainly directed for in class use and to provide a high level view on it. For instance, if a teacher notices any problems (e.g., the successive amount of red rectangles) while the pupils are in the classroom, he/she can approach them and provide immediate support.

Progress Matrix

When monitoring students, having a notion of each student's overall progress is important. By knowing which type of concepts the students already master, teachers can tailor their feedback to be more specific for each student's case. The progress matrix displayed in figure 2.5 tries to give this notion of progress to the teachers.

Each square in this matrix indicates the 'status' of the most recent submission received from a single student (YY axis) for a particular assignment (XX axis). The 'status' of the submission is color coded, as is customary.

(a) Live Feed page (b) Progress Matrix page

Figure 2.5: Teacher's view - CodeInsights visualisations

2.4.4 Student View

The student's primary role in CodeInsights is to provide information to the system via their source code. By allowing instructors to have access to better information about their work, they benefit from better support. Despite being designed to support the decision-making of instructors, students were not disregarded. For this reason, a lot of information and visualization are mainly located in the teacher's view. Nonetheless, students have access to some important features.

Starting with the IDE plugins, they function as a means to send information to the server. Although their main objective is sending data they can also receive information from the server. As each student sends a submission to the server, the server will send back a response with test case results. It basically indicates in which test cases the student

managed to pass or fail. In addition, if the test case was classified as partially valid (i.e., the test case output is matched in a high percentage) the student is reminded to verify their output printing method. Figure 2.6 shows the type of feedback students receive when programming in their IDE.

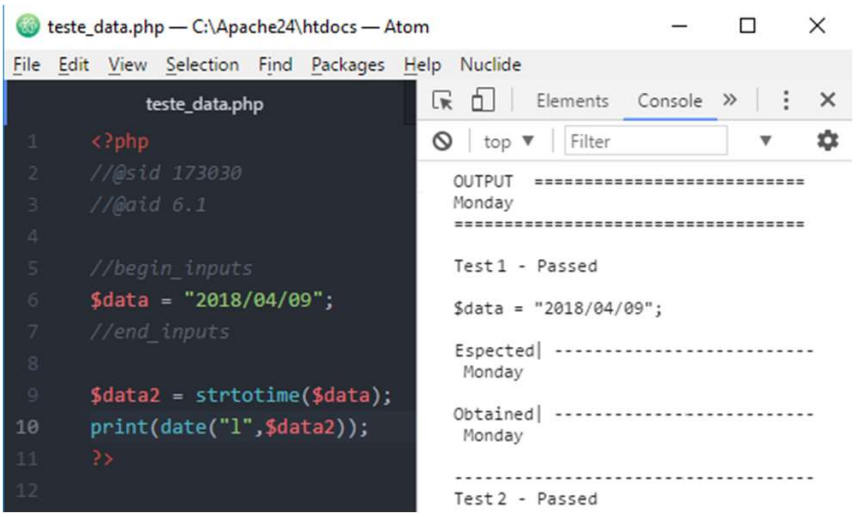


Figure 2.6: Example of the test case results seen in the IDE plugin

Like the teachers, the students have access to a web interface of their own. A single web server (Apache) is responsible for handling the requests made by both students and teachers. The student´s side and the teacher's side of the web applications are delimited and separated by different URLs (Uniform Resource Locators). Teachers utilize the 'standard' path *http://server_ip/CodeInsights* and students use the sub-path *http://server_ip/CodeInsights/students* ensuring there is mixing up between these two entities.

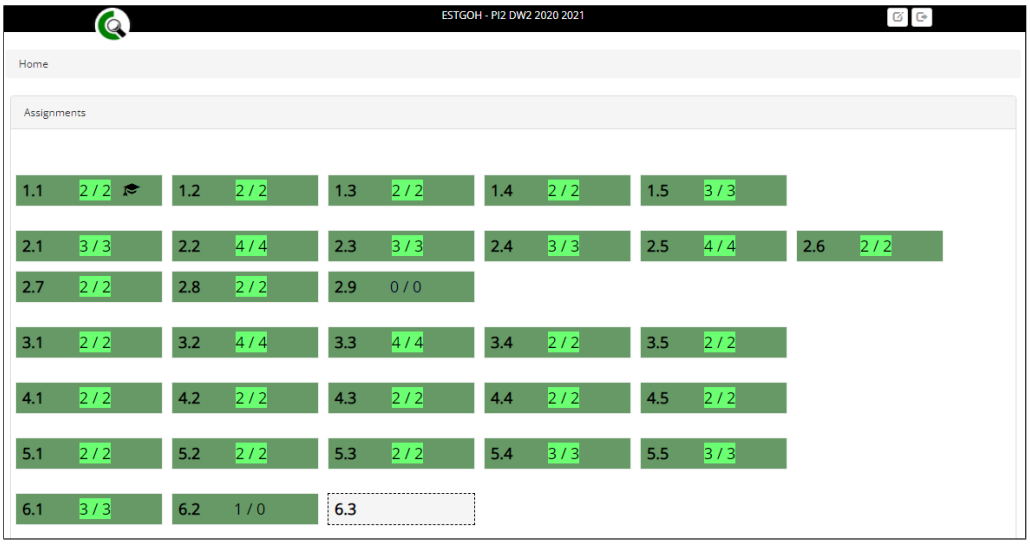


Figure 2.7: Student's view - Home page

Students can enter their part of the web application by inserting a set of preemptively defined credentials. Once a student presents the system with the right set of credentials they are shown their overall progress. Figure 2.7 shows how the student's view main page is organized. On this page, we can see that all the assignments grades are displayed. In

a similar fashion to the teacher's views, this main page utilizes a special color scheme to represent the classification of each snapshot.

Having a notion of which assignments were done is important for students to locate themselves, and if need be revisit some of their previous difficulties and solutions. For this purpose, students can see their last submission of each assignment by clicking on its respective rectangle on the home page. Besides viewing and analyzing their submitted code the system allows them to edit it.

2.4.5 Performance

Until now we have explained how source code is collected and how teachers have immediate access to it in the form of different visualizations. With these visualizations teachers can already identify outliers in the student's performances. But this 'manual' analysis would still require for the teacher to be using the system, which goes against the whenever, wherever principle of this system.

In order to provide information on which students are facing such problems, CodeInsights uses the concept of performance. To calculate and quantify each student's performance, the system uses a simple division to obtain the percentage of assignments completed. Besides the number of assignments completed, the teacher can add weight to each assignment to simulate the variety of difficulties that assignments might have.

Each student's performance is then compared to the class's overall performance. If a student has performance superior to the class mean plus the standard deviation, it is qualified as a 'fast paced' student. On the other hand, if the performance is inferior to the class mean minus the standard deviation, it is categorized as a 'slower paced' student. Figure 2.8 illustrates how fast and slower passed students are identified.

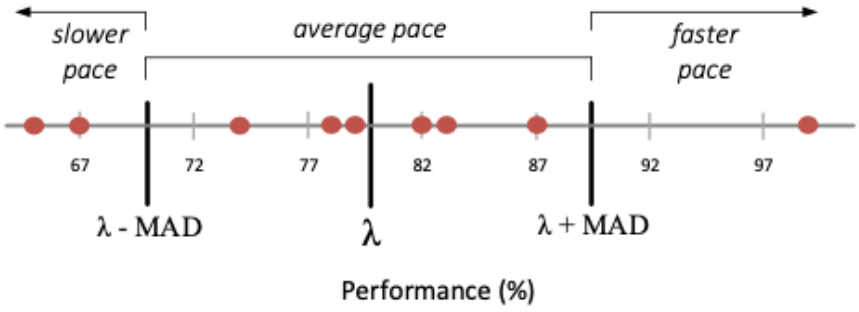


Figure 2.8: Illustration of the performance calculation approach, taken from [20]

After slow and fast paced students are identified the teacher is notified of this event. Another important event is the increase or decrease in performance, as it may identify changes in the interest of students. Notifications play a big part in bringing critical information to teachers without requiring them to be actively using the system. The following section will go over how to create notifications and how they work.

2.4.6 Notifications

The fact is that teachers have a lot of work in their hands for the little time they have. One of the main attributes of CodeInsights is to be able to provide critical information regardless of the time or place. To do so, instructors have access to a wide range of notifications, like a high number of submissions after the due date, a large number of unfinished assignments, an unusual amount of lines in a submission, and many others. Every notification is sent to the teacher's email and if possible is displayed as a pop-up on the screen.

Notifications can be classified as online or offline. Online notifications are generated in real-time (i.e., as the snapshots are processed). On the other hand, offline notifications are created by the system at a designated time for example, once a day. Additionally, offline notifications are characterized by their broader meaning, such as an unusual number of students unable to solve a certain assignment correctly.

In order to notify the instructor of the occurrences of certain events while not overwhelming them with too much information, a cognitive attention system based on surprise is used. Macedo's model [19] is built on three elements: i) Surprise Value of Information, (ii) Uncertainty-based Value of Information and (iii) Motive Congruence/Incongruence-based Value of Information. An event is considered valuable and so displayed to the instructor if it is a desired event and it is surprising or it represents a considerable amount of information gain. To tailor this system to the instructor's needs, he/she can customize each one of these three elements thresholds.

2.4.7 Plagiarism

Plagiarism is a major problem all trough-out education, universities are no exception. According to a survey [8] conducted to analyze the student's perspective on plagiarism, as many as 72 of the software students participating looked at previous year's assignments to get 'inspiration'. Some of the reasons attributed to these inspirations were: the time pressure, the student's interests, or even the teacher's approach. However, one of the most impactful factors in plagiarism is poor academic skills [13]. For example, a student who is facing a problem might be tempted to just ask a friend for his code instead of asking the instructor for guidance.

Usually the detection of plagiarised code is made after some sort of formal submission. Since CodeInsights uses a different perspective (i,e real-time submissions), it can detect plagiarism during the entire student's programming process, not only at the end. By extending the instructor's time to address plagiarised code, the system allows instructors to impede the evolution of copy and pasting code into higher levels of plagiarism [17].

To verify the presence of plagiarised code, CodeInsightrss compares each new submission to every last submission made by other students. Since its objective is to provide real-time information about plagiarism, this process should take as little time as possible. Thus the comparison between each submission is made by using a simple lightweight string comparer. This algorithm [17] produces a similarity value allowing the teacher to define a minimum threshold value, for which a snapshot is considered as plagiarized from other snapshots of the same assignment.

Chapter 3

Related Work

We covered the CodeInsights system in great depth in chapter 2, a program dedicated to assisting teachers in their activities. This chapter is in charge of framing the system in question with the programming monitoring tools' environment. Section 3.1 provides background information on the many types of tools utilized in the learning environment. In section 3.2, we take a look at some of the most well-known monitoring tools and highlight some of their main attributes. Finally, we present all the main differences between these systems in an aggregated manner in section 3.3.

3.1 Background

In the old days [1] computer programs were only assessed statically (i.e., without actually running the program). Normally a student would deliver printouts of the program code and output, and teachers based their assessment on visual inspection. Static assessment revolves around the analysis of the code structure and its design; it relies on the syntactic and semantic correctness of the program. Even though statically assessments can provide a lot of useful information they can only do so much, as the major attribute of code functionality is mainly untouched.

As time went on technologies evolved and internet communications became a standard way of exchanging data. Printouts were no longer required, and electronic submissions became the norm. Teachers could effectively run the students' program on their machine and test its behavior. This way of evaluating code is referred to as a dynamic method of assessment (i.e., one that actually requires running the program).

In order to automatically analyze the behavior of a program and its correctness, automatic assessment tools were created. Teachers used this type of tool to facilitate the grading process, which until then was rather manual. Usually, automatic assessment tools resort to test cases to validate the functional component of a program. This dynamic form of assessment runs the code against a set of test cases and compares its textual outputs. The process of automatically classifying the students' code is normally denominated auto-grading and it plays a major role in systems that collect students' snapshots.

Automatic assessment tools can be used to help instructors in grading assignments, but are mainly directed to supporting students' programming process with feedback from its test case results [1]. Lately, there has been a shift in the perspective of assessing programming work, as the instructor becomes the main focus of support. These tools are denominated

monitoring tools, like CodeInsights, they try to provide the teacher with valuable information about their student's work and progress, not just grading. As a result of this shift, there is not currently a wide range of monitoring tools available.

Some of the earlier monitoring tools [33] used these plugins to record code changes in an XML format. Upon collecting this information teachers would utilize a special tool designed to convert the XML to excel, where teachers could visualize a set of graphics about their students' code. Since then we have come a long way and with the right approach and right technology, it is possible to eliminate a lot of the overhead present in these conversions of data formats.

3.2 Similar applications and services

A state of the art [47] describes the current knowledge about the studied matter through the analysis of similar or related work. It provides a comprehensive overview of what has been done in the field and what can be further investigated. In software development, since systems are specially designed for users, it is a really common way of analyzing the 'competition'. Normally this search is done prior to the development of a system, to provide a foundation on ways a tool can differentiate itself from all the others already present in a particular ecosystem.

Since CodeInsights is an already functional tool, it would not make sense to do a 'state of the art' style of evaluation on the monitoring tool environment. This step was already performed during its conception and provided ways for the tool to distinguish itself (e.g., its real-time properties). However, for contextualization purposes some of the most relevant names in the monitoring tool environment are presented in the following subsections. Each sub-sections will only briefly describe each system and its main attributes.

3.2.1 ClockIt

ClockIt [35] is a set of tools that instructors can use to analyze their students' working habits and programming techniques. This system can be divided into two parts: data collecting and data visualization. The collection of data, specifically java code, is done through a BlueJ plugin, an integrated development environment (IDE). Due to its smaller and easier interface, BlueJ is primarily used by novices. The BlueJ data-collection extension captures particular events of interest (e.g., compile events, package events, and invocation events) and registers them into a logfile located at the project directory. Every time the project is closed this file is submitted to the main server for data visualization purposes.

The data visualization is supported by a web interface that displays charts such as most common errors, the evolution of the project size, the number of evocations of each method and the average amount of time each student spends on a project. Even though these types of metrics can be very useful to teachers, they can also provide crucial inside knowledge to students.

With students in mind, the system allows them to also access the web interface component, even though with limited access (i.e., to their own related data). Considering that all the data is stored in each student's project directory ClockIt extends, yet again the IDE functionalities and creates a series of visualizations with another ad-on. In reality, there are little differences between the IDE or web interface approach, only that while using the

IDE students are restrained to seeing metrics to the project they have currently open.

In essence, whereas CodeInsights provides an auto-grading component and subsequent real-time notification about specific events, ClockIt analysis mainly resorts to graphics.

3.2.2 Retina

Retina [34] is a system that collects information about the student's programming activities. This information could be sent via a command-line compiler or an IDE plugin. A modified version of a command-line Java VM is responsible for capturing runtime errors, while the IDE plugin captures compilation errors. Independently of its nature, all data is stored in the server's relational database. The instructor can access the collected data by either using a standalone desktop application or by using a web browser interface.

Teachers can examine students' specific information. The student's compilation errors and the time student one spends on their assignments are some of the principal metrics captured by Retina. Additionally, the system supports a class view that gives a broader perception of the class's progress, enabling teachers to address some of the most common errors students face.

This tool puts an extra effort into the student's view by providing information about their most common errors and making some suggestions based on previously recorded semesters. Another important trait of Retina is the ability to make real-time recommendations to students based on their programming activities. Retina's Recommendation Tool works with popular chat systems, namely: Yahoo! Messenger, Windows Live Messenger, and Google Talk. Users can 'subscribe' to this system by initiating a chat session with Retina. After being successfully identified, students will be recommended to take certain actions as the tool sees fit. For example, if a student is having a lot of compilation errors (i.e., at a higher rate than the class's average), the tool will suggest the student to work in smaller intervals. Retina does not approach the code's validity, so it is missing the grading component, unlike CodeInsights.

3.2.3 SCORE

Score [51] is a software package that allows the capture and analysis of the students coding processes. Like normally, the tool utilizes an IDE plugin to capture information while students are programming. The information captured is then used to help teachers identify arousing problems with the help of a dedicated analysis tool. Score takes a different approach on what information it mainly captures and how it exchanges that information between the capture and analysis component.

Firstly and starting with what type of information recorded, the NetBeans plugin uses a code text approach rather than an output based one. This means the plugin tracks textual changes in the students' source code. Typically, to acquire information, these programs rely on output comparison of code, where they record error messages or output by running a set of unit tests. The score's IDE plugin data structure is represented by different entities who represent different levels of alterations made to the project. At a broader scope, the add-on is capable of recording project file changes, for example, the addition or deletion of a class file. File changes are also represented by their modifications, more specifically the alteration of source code lines. Various mutations to the lines of code form line histories. *Line history* and *Files changes* form the basic alterations captured and therefore used in the visualization component.

Secondly and addressing the method of data exchange, unlike the usual approach where these plugins follow a client-server philosophy, this add-on utilizes a local storage model. This means that the various versions and changes to the source code are recorded in a subdirectory of the project. As a result, when students submit an assignment using a conventional LMS, the information generated by the plugin will be stored in a subfolder of the project.

The source code analysis tool is composed of a variety of features, some of which need to be referenced. The basic strategy when evaluating the students' code is done with a common diff-style view, comparing all of the changes (i.e., additions, deletions and modifications) made to the files of the project. The instructor can be more particular and do a line history analysis that displays a list of all *Line Histories*. For each *Line History*, a list of all mutants (i.e., their alterations) is displayed.

Visualizations play a major role in monitoring tools and help teachers visually understand some of the system metrics. Score is no exception and allows the teachers to see file changes and line histories in a graphical manner. For example, it is possible to see which line was the most problematic in a project, by checking a graphic that maps the number of changes made in each line. Another key attribute of Score is the ability to create a screen capture for each project version, making this tool especially useful in Computer Graphics courses by visually displaying the student's progress. Finally, the tool utilizes a categorization system, where teachers can take notes in certain line histories or file changes. Teachers are also able to group the code alterations into specific categories, which are then used in conjunction with the system visualizations.

3.2.4 TestMyCode

TestMyCode [50] is an automatic assessment system dedicated to removing some of the manual steps needed to evaluate the students' programming work. This tool follows an extreme apprentice approach which is reflected in many of the system's functionalities. Extreme apprentice is based upon the concept of extreme programming, which is a software development methodology that values feedback and communication. The computer Science course's main goal is to prepare the students for the working environment, so it is crucial to implement some of the industry's practices (e.g., Extreme apprentice) in the teaching process.

The system has a data gathering component that is supported by an IDE plugin, as it is common in snapshot capture solutions. When students save or compile their code, TestMyCode uses a NetBeans IDE plugin (the only one supported) to transmit source code to sandbox servers. Each sandbox server features an optional Maven cache for storing library dependencies (e.g., for web development courses), which is a unique feature of TestMyCode. The IDE plugin operates in a minimally intrusive manner to the students' programming and provides some extra features. Students have direct access to course exercises in their IDE, meaning there is no need to utilize pdf worksheets to see the problem descriptions.

Each submission is tested against a set of test cases (auto-grading) defined by the instructor, who can then manually review each code submission. Manual reviews play a big role in TestMyCode and its extreme apprentice approach by allowing for feedback to flow bidirectionally between the two entities involved in the learning environment. On a web browser, teachers can decide if some students need to readjust their approach to the exercise by doing code reviews and sending them feedback. Alternatively, a student can ask for a code

review directly on their IDE plugin. Code reviews are a common practice in the industry since they improve the quality of written code. So by facilitating and proposing frequent code reviews TestMyCode enables errors to be fixed in the earlier stages of development when they are easier to correct. Besides manual reviews, the system uses dedicated tools to make this process more automatic, such as Findbugs [36] for Java.

Scaffolding principles, or gradual enhancements to exercise tasks, are used by TestMyCode when creating assignments. Each assignment will start with a predefined template that helps direct the student away from bad habits. For example, the teacher will supply the students with all the necessary code except for a method implementation, leaving this task to the students. By doing it this way, the teacher makes sure methods are correctly named and that students make use of the functions modularity propriety. After that, tasks will gradually build upon the previous ones and the student can only advance if it successfully programmed the previous task. Each one of the tasks has its own set of test cases and has a classification of its own, once again re-forcing the idea of the incremental nature of programming.

3.2.5 Spinoza

Spinoza [11] is a web-based tool-set developed to support active learning. It consists of a web based IDE where students can submit their code and a web application where instructors can create their assignments and challenges. The creation of unit tests supports the creation of assignments. Resembling CodeInsights, this approach makes Spinoza able to qualify student code and provide real-time feedback. This system differentiates itself by using a Think/Pair/Share pedagogy where students with similar code are grouped. The usage of an IDE, although not mandatory, is pretty useful since it does not break the student's workflow. This lack of IDE support is something lacking in Spinoza.

3.3 Summary

As stated in the preceding section, tools that assist teachers in monitoring their students' code might have a variety of qualities and attributes. To summarise each one of the described system's main features, table 3.1 shows a direct comparison between each system's qualities.

In essence, the attributes taken into account when comparing all the monitoring tools were: the presence of auto-grading, the set of supported languages, the real time capture of code, the use of visualizations, the use of notification and the type of errors captured.

As can be seen, CodeInsights stands out from the competition due to its high level of diversity. To begin, we should point out that certain systems are fairly limited in terms of the types of errors they can detect (e.g., the ClockIt IDE plugin is only able to record compilation errors). Furthermore, CodeInsights has a wide range of supported programming languages (i.e., it supports practically every language except C++). Where some systems rely on a local model (e.g., Score records code changes in a subdirectory of the projects to be manually delivered to an LMS), CodeInsights can retrieve data in real-time with a server-client approach. Additionally, the majority of the systems reviewed only deliver information through visualizations, which still requires a significant amount of manual work in identifying students in need of help. CodeInsights, on the other hand, includes a notification system that notifies instructors of key events without forcing them to use the

system. Even though it is not listed in the table, we must note that CodeInsights offers a means to identify plagiarism which is not normally done.

Tool / **Attributes**	**Score**	**Retina**	**Test My Code**	**CodeInsights**	**ClockIt**	**Spinoza**
Autograding/Unit testing	X	X	✓	✓	X	✓
Teacher visualizations	✓	✓	X	✓	✓	✓
Real Time	X	✓	X	✓	X	✓
Student Feedback	X	✓	X	✓	X	✓
Notifications	X	X	X	✓	X	X
Suported Languages						
Java	X	✓	✓	✓	✓	✓
Python	X	X	X	✓	X	X
PhP	X	X	X	✓	X	X
JavaScript	X	X	X	✓	X	X
C	✓	X	✓	✓	X	X
C++	✓	X	X	X	X	X
Type of Errors						
Run Time	X	✓	✓	✓	X	✓
Compilation	✓	✓	✓	✓	✓	✓

Table 3.1: Comparison - CodeInsights vs Popular monitoring tools

Chapter 4

Requirements

Requirements engineering is a specific branch of software engineering concerned with the identification of a software's purpose and in which context it will be used. This activity is formed by several tasks like requirements elicitation, analysis, documentation, validation and management. A requirement is a way of describing and communicating to the client what the system must do. Typically requirements are split into two types: Functional and Non-functional requirements. In essence Functional requirements describe *'what'* the system must do, while the Non-functional describe *'how'* it does it.

Following a requirements engineering approach, this chapter describes the whole process of translating the client's needs into requirements. In section 4.1 we describe what were the new functionalities for this tool. Section 4.2 states some of the system's quality attributes or in other words its Non-functional requirements. The last section of this chapter: section 4.3 explains the different types of constraints to which our project was subject.

4.1 Functional Requirements

Functional requirements define the features that allow the system to work as intended. They specify the functions a program must complete (i.e., its behavior to certain inputs and its expected outputs).

CodeInsight's Functional Requirements (high-level) were gathered through meetings with the project creator, who acted as a client. These meetings were also crucial in terms of validating them. In addition, the data gathered during the monitoring tool environment analysis was taken into consideration. Once we had a general idea of the features needed, we started breaking down each Functional Requirement into a series of finer detailed Functional Requirements. Now that the Functional Requirements election and analysis were concluded we had to document the outcome of these processes.

There are several ways of documenting Functional Requirements, the most popular and often used are user stories and use cases. We decided to describe the system's functionalities as use cases. Our increased understanding of the use case format and the precise details it provides for the development phase were the driving factors for choosing this method. Each use case followed the template provided by CockBurn [9], with some slight alterations (i.e., the deletion of some of the template's sections) to provide the right amount of detail without becoming too 'heavy'.

Category	ID	Use Case	Priority
Entities	1	Managing students in the system	High
	2	Managing teachers in the system	High
	3	Managing classes in the system	High
Edition	4	Edition managment	High
	5	Edition selection	High
Passwords	6	Forgot password	High
Assignment	7	Importing exercises (Different editions)	High
	8	Inserting assigments (Test cases)	High
	9	Rich text description	High
System Integration	10	Data import	High
	11	Data export	High
Dashboard Customization	12	Deleting dashboard graphics	High
	13	Adding graphics to the dashboard	High
	14	Changing the layout of graphics on the dashboard	High
	15	Changing the size of graphics to dashboard	High
Processing	16	Accept processing code	Medium
	17	Display processing code	Medium
Chat	18	Starting a chat	Medium
	19	Sending a message via chat	Medium
	20	Chat Selection	Medium
Share	21	Create a code sharing room	Medium
	21	Share code	Medium
	23	View all of the share rooms	Medium
Segurança	24	Protection against malicious code execution (System)	Medium
	25	Adding keywords to a teacher's blacklist	Medium
	26	Protection against malicious code execution (Teacher)	Medium
Misc	27	Code peer-reviewing	Low

Table 4.1: Functional Requirements - Use cases

Table 4.1 represents an overview of all the new functional requirements of the new version of CodeInsights and their use cases. Due to a large number of Functional Requirements, a document detailing use cases and an explanation on why they were added to the system was created. This document denominated *Functional Requirements* is also annexed with this thesis in Appendix B.

Of all the use cases present in table 4.1 we must note that some were not initially planned. The sharing code and forgot password use cases, with id 6 and 21-23 respectively, have been added since the writing of the intermediate report.

To ensure that clients' requirements are met, software developers prioritize their use cases so that those which are most important are developed earlier in the development of the project. The correct prioritization implies that if some problems surge and delays appear, the functionalities with the most priority will be implemented first.

Since the project creator was a member of the team, he voiced his thoughts on the most important features that should be incorporated first. As a result of his feedback, each use case was assigned a level of priority. The learning curve needed for utilizing new tools was also taken into account in the prioritization process (e.g., the CRUD operations were prioritized with the highest value). Priority is a metric that determines how important a function is: high, medium, or low. High priority means that the feature must be included in the next version or the system will be rendered worthless. A feature should be provided as quickly as possible if it is given a medium priority. Low priority means that it would be nice to have at some point in the future.

4.2 Quality Attributes

A Non-functional requirement defines the quality attribute of a software system [38]. Before we can describe non-functional requirements, we must first define quality. W.Edwards Deming [6] defined good quality as "a known degree of uniformity and dependability with a quality standard suitable to the customer". Thus we can conclude that non-functional requirements are a collection of standards used to evaluate a system's specific operation while always keeping the client's demands in mind.

It is of utmost importance to consider these system qualities throughout a project's life cycle, but especially during its design as they directly impact the system's architecture. Yet again Nuno's (creator of CodeInsights) insight of the system provided a crucial perspective in identifying these desirable properties for the system.

Each quality attribute of CodeInsight was described as a scenario in the conventional six-part format. Figure 4.1 shows the parts of a quality attribute scenario. The following subsections the most relevant quality attributes and their scenarios.

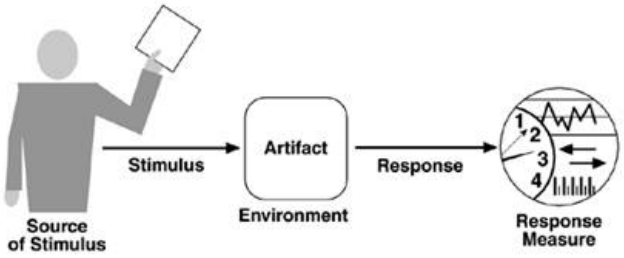


Figure 4.1: Quality attribute parts taken from [37]

4.2.1 Performance

Performance is the measurement of how long it takes the system to respond when an event occurs [37]. With the new version of CodeInsights, we were especially interested in the code submission event, since during a specific period (e.g, the minutes before an assignment deadline) numerous concurrent students might be programming.

Stimulus:	The student saves his code in an IDE submitting it to the system
Source:	Student
Artifact:	CodeInsights, more specifically the Sandbox component
Environment:	During a stress time for the system, with multiple users at the same time
Answer:	The system runs the students code and return its output
Metric:	The system will have to perform this operation in less than 3s, not being noticeable to the user

Table 4.2: Performance - Scenario 1

Additionally while in the performance context, we taught the delivery of the chat messages and the sharing code messages should also be taken into account. Essentially sharing code demands higher responsiveness, whereas help requests sent via a chat could take longer to be delivered.

Stimulus:	The teacher writes some source code in the sharing code page
Source:	Teacher
Artifact:	CodeInsights
Environment:	During normal runtime
Answer:	The system receives the code and delivers it to the student
Metric:	The system will have to perform this operation in less than 1s, not being noticeable to the user

Table 4.3: Performance - Scenario 2

Stimulus:	The teacher send some message containing feedback to the student
Source:	Teacher
Artifact:	CodeInsights
Environment:	During normal runtime
Answer:	The system receives the message and delivers it to the student
Metric:	The system will have to perform this operation in less than 30s

Table 4.4: Performance - Scenario 3

4.2.2 Security

Security is the measure of the system's ability to withstand unauthorized attempts to use data or services, while providing access to legitimate users. It consists of three sub-quality attributes: confidentiality, integrity and availability. We were especially concerned about the sandbox server component's integrity and confidentiality for the new edition of CodeInsights as it runs 'unknown code', and we can't assume every student has the best intentions.

Integrity

Integrity is the property that data or services are being delivered as intended. With the improved version of the system we wanted to deny the ability to alter the system by running some sort of malicious script.

Stimulus:	The student attempts to execute malicious code to alter the system data by submitting a snapshot
Source: **Artifact:**	Student CodeInsights, more specifically the Sandbox component
Environment:	In any situation that CodeInsights finds itself in
Answer:	The system will not allow the execution of such malicious code, afterwards notifying all stakeholders
Metric:	1.The student is able to execute malicious code 2.The student is not able to execute malicious code

Table 4.5: Integrity scenario

Confidentiality

Confidentiality is the property that data or services are protected from unauthorized access. Since the main-server and the sandbox server coexist confidential information is stored in the same machine. With the new iteration of CodeInsights we needed it to be able to deny access to such information.

Stimulus:	The student attempts to execute malicious code to access other students data
Source:	Student
Artifact:	CodeInsights, more specifically the Sandbox component
Environment:	In any situation that CodeInsights finds itself in
Answer:	The system will not allow the execution of such malicious code, afterwards notifying all stakeholders
Metric:	1.The student is able to execute malicious code 2.The student is not able to execute malicious code

Table 4.6: Confidentiality scenario

4.2.3 Internationalization

Software internationalization refers to the product's property to be localized for target audiences that vary in culture, region, or language. For the new version of CodeInsights we were particularly concerned in allowing other languages besides English to be utilized.

Stimulus:	Some teacher require a CodeInsights installation with a specif language in mind
Source:	Administrator
Artifact:	CodeInsights
Environment:	During CodeInsights deployment
Answer:	The system will allow the support for multiple languages
Metric:	The administrator is able to easily switch the languages of the system

Table 4.7: Internationalization scenario

4.3 Constraints

By definition, a constraint is a state in which the freedom of action is severely restricted. Every software development project encounters several constraints that are highly influential architectural drivers. These constraints limit design choices since they are fixed and can not be overlooked [24]. While functional and non-functional requirements can and should be negotiated with a client, one could see constraints as a 'must-have' requirement. As such, they must be taken into account before designing a project's architecture.

These constraints can be one of two types: Business or Technical. Technical constraints are technical restrictions that are made during the system design process. Usually, these constraints are provided by the stakeholders of the project and in software development are highly related to the development technologies. Business constraints on the other hand instead of focusing on technology are rather related to business decisions. A business

decision is a decision taken by a company in order to meet its needs and expectations. As a result, these limits are frequently articulated in the form of a budget and/or a schedule.

When designing the new system, there were three key constraints imposed by external factors. Since CodeInsights is a fully functional system, its previously used technologies were our major technical constraint for the proposed work. The two other constraints fell into the business category.

The design must support the previous system technologies
Seeing that a lot of work and time has been put into this system, even though it would be possible, it did not make sense to start using other technologies. So the development was restricted to using PHP as a back-end language and HTML/JavaScript as front-end technologies.

The design must be implemented until 31 of October
Like all projects, its schedule is a vital part that acts as a limiting factor. The project had until 31 of October to be delivered. Planning is further discussed in chapter 7.

The design must be implemented within 38 weeks, with a weakly 40 hour budget
Normally, a project's scope is constrained by its budget allocation. This measure makes little sense in our project's context, so we defined it in hours instead. In essence the project had a ceiling of more-less 1520 hours.

Chapter 5

Design

The new functionalities, quality attributes and constraints described in chapter 4 form the so-called architectural drivers. To satisfy the architectural drivers this chapter outlines which system components were added or modified. There are virtually countless possibilities and options to achieve the designed requirements, so we will explain our choices and their rationale in section 5.1. We also take this chapter to define some of the usability guidelines we took when designing the system mockups in section 5.2. In the following section 5.3 we present a mockup prominent from this designing phase. Finally, section 5.4 offers a summary of all changes made to the system architecture, particularly the addition and modification of components.

5.1 Components of the final architecture

With all the requirement-related processes now fulfilled, and before advancing to the implementation phase, it is good practice to first design the system architecture. So we started evaluating the different ways to achieve the desired system requirements. All of the design decisions made in the subsections that follow are primarily concerned with the functional aspects of the requirements. Although in lesser quantities, quality attributes and system constraints had an impact on system architecture design choices.

5.1.1 Forget Password

The forget password is a critical component, and at first sight may come as a straightforward operation, but due to the nature of the information being addressed, it is rather not. The Open Web Application Security Project (OWASP) is an online community that produces freely available articles and methodologies in the field of software security. OWASP provides a set of guidelines to help protect the forgotten password against some popular attacks [40]. The basic guidelines to protect the forgot passwords service are as follow:

- Return a consistent message for both existing and non-existent accounts.
- Ensure that the time taken for the user response message is uniform.
- Use a side-channel to communicate the method to reset their password.
- Use URL tokens for the simplest and fastest implementation.

- Ensure that generated tokens or codes are:
  - Randomly generated using a cryptographically safe algorithm.
  - Sufficiently long to protect against brute-force attacks.
  - Stored securely.
  - Single use and expire after an appropriate period.

When designing CodeInsight's forgot password mechanism we made sure to take all the OWASP security notions into account. Next we will describe the six main attributes of our designed forgotten password mechanism: Communication Channel, User Validation Method, Token Attributes, Token Validation, Token Hashing and User Feedback. In addition, a flowchart that details every step of the forgotten password mechanism can be found in Appendix A.

Communication Channel
Firstly it was decided to use a mailing service as a side-channel to transmit the password-reset procedure to the users. Normally these kinds of mechanisms send information via mail or SMS, but since CodeInsights already possessed a mailing service configured to provide feedback to the students, the choice was pretty natural.

User Validation Method
The next decision to be taken was how to validate the user in a secure manner (i.e., to know if the user is in fact who he claims he is). As said by the OWASP entity, the easiest way to achieve this is via URL Tokens. The basic steps to take when using URL Tokens and the ones we followed were: **1)** Tokens are generated once users request new passwords and should be stored in the system database. **2)** Once a token is created it is then directly attached to an URL and sent via email to the user who needs to replace his password. **3)** The user should receive an email with a set of instructions to follow. **4)** Once he follows the instructions and clicks the link with the correct URL attached he is redirected to a page where he will define a new password. **5)** On this page, the token present in the URL is validated (i.e., if the token in question matches the one in the database), and then the user can provide the new password. **6)** Two equal passwords must always be required to prevent typos. **7)** Once the password is reset all it is left to do is for the user to attempt to login into the system as usual. **8)** The user must not be directly logged in to the system when the password is reset, as this adds to the authentication complexity and increases the likelihood of introducing vulnerabilities.

Token Attributes
The tokens generated by the system also followed some restrictions in order to provide greater security. To prevent brute force attacks (i.e., the repetitive attempt to guess the token in order to access the system) the token size is 24 characters. By choosing 24 chapters we made sure the number of different possibilities is large enough it would take an attacker a lot of time to guess the correct token. For ease of use and to avoid problems with the URL, tokens are base62 encoded (A-Z a-z 0-9).

Token Hashing
Hashing consists of a mathematical algorithm that maps data, in our case a 24 character token is mapped to a bit array of a fixed size. Hashing is a one-way function, that is, a function for which is practically infeasible to invert or reverse the computation. By deciding to hash a token, even if the database information is compromised by another assault, such as a SQL injection, the attacker will only be able to find hashed tokens, not their original plain text values. Normally in these types of algorithms, there is a trade-off

between performance and security. In general, the longer an algorithm takes to hash a token, the more secure it is because the attacker has less time to brute force the system. We choose BCRYPT, a cryptographic algorithm that aims at being slow and that has a good reputation since it is yet to be broken. By being slower it is more secure than some other popular algorithms, like SHA-256 which can be very efficiently implemented on a GPU leading to more password tries per hour.

Token Validation
On top of all these security measures, if by mere luck an attacker managed to reverse the hash value to its original token, we attributed each token a validity date. A token is only considered valid for about twenty minutes, after that it becomes revoked and unusable by a user or attacker. Since the process is relatively fast, twenty minutes are more than enough. For convenience, the recipient of the email is always informed the link expires and how long he has until the link expires.

User Feedback
Finally and as suggested by OWASP the feedback provided when attempting to reset a password of an email, was consistent no matter the case. Whether the email was associated with a username on the system or not, the message provided by the system was always the following: "If that email is registered in the system, a set of instructions on how to reset the password will be sent". This meant an attacker could not gather information on which emails were registered in the system.

5.1.2 Processing

Processing [18] is a Java-based programming language that enables users to code in a visual arts context. It was created from the ground up with the goal of providing immediate visual input. We decided to support this language since it is usually used in introductory programming due to its visual attribute that makes the learning process 'lighter'.

Because CodeInsights is a web application, it is difficult if not impossible for teachers to observe Java code executing in their browsers in real-time. Every web browser has a JavaScript interpreter built-in, which means that JavaScript programs can be run on web browsers. Fortunately, there is a direct port of the Processing language to JavaScript denominated p5.js [52]. In essence, Processing is an environment based on the Java programming language while p5.js is a library based on the JavaScript programming language.

Language Translation
Simplifying it, for CodeInsights to support the Processing language we needed to translate Java code to JavaScript code. Despite their similar names Java and JavaScript are not remotely related to each other, making the translation process have some caveats. The organization responsible for the creation of p5.js provides a set of basic rules to follow when translating code [21].

In the whole translating process there are some direct changes, many of which are methods and variable names which follow different denominations. For example in p5.js the method *size()* has been replaced with *createCanvas()*. We can achieve this change with some simple regular expressions that match this function name and replace it with the correct one.

Other changes are far more complicated and require the use of some frameworks. In p5.js you need to assign global variables values inside the *setup()* function. In order to achieve this, we have to understand what global variables are and the difference between variable declarations and their assignments. A global variable is a variable that is declared in the

global scope, in other words, a variable that is visible from all other scopes. The process of variable declaration registers a variable using a identifies and attributes it to a specific scope, whether that would be global or not. The process of variable assignments is the attribution of a certain value to the variables previously declared. Oftentimes the process of declaring and attributing a value to a variable is done simultaneously, but with p5.js this process has to be done separately. Figure 5.1 depicts the source code translation needed for a basic Bézier curve program in Processing.

```
1  // before the translation
2  int canvasHeight = 640;
3  void setup() {
4    size(canvasHeight, 360);
5    stroke(255);
6    noFill();
7  }
8
9  void draw() {
10   background(0);
11   for (int i = 0; i < 200;
         i += 20) {
12     bezier(mouseX-(i/2.0),
         40+i, 410, 20, 440,
         300, 240-(i/16.0),
         300+(i/8.0));
13   }
14 }
```

```
1  // after the translation
2  var canvasHeight;
3  function setup() {
4    canvasHeight=640;
5    createCanvas(canvasHeight
       , 360);
6    stroke(255);
7    noFill();
8  }
9
10 function draw() {
11   background(0);
12   for (var i = 0; i < 200;
         i += 20) {
13     bezier(mouseX-(i/2.0),
         40+i, 410, 20, 440,
         300, 240-(i/16.0),
         300+(i/8.0));
14   }
15 }
```

Figure 5.1: Converting Processing to p5.js

AST

In order to understand how we can manipulate the code and systematically make these changes we need to understand the concept of code compilation. Code compilation, or the translation of human readable code to program readable code, is normally divided into: Tokenization, Parsing and Code Generation. To manipulate code, the steps of tokenization and parsing are really important as these stages output a well defined structure that can be altered with ease and consequently change the source code itself. This structure is denominated abstract syntax tree, or AST for short.

With an AST we can decide what tree nodes to move, delete, add or even update. For example, it is possible to change the scope of some variable declarations like the need expressed above for p5.js to work properly. In the end, since parsers (i.e., the programs responsible for parsing code) are highly complex and require a lot of work, we choose an already built PHP library called Peast (PHP ECMAScript Abstract Syntax Tree) to generate and manipulate the JavaScript AST trees [32].

5.1.3 Database

CodeInsight's relational database serves as the system's heart, storing all of the data required for it to function properly. A relational database [43] is a collection of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows. Each row in a table holds information about a certain entity where each of its attributes is represented by a column. The tables and their relationships are a reflection of the system design to accommodate its desired functionalities.

The previous database structure was built on a variety of assumptions that have changed over time. Some database entities were updated and others were created to achieve the system's new features, but there was an effort to retain the prior schema as close to what it was as possible, only building on top of it. By maintaining the database structure as similar as possible, the changes were kept to a minimum in order to reduce the risk of error. The table addition sometimes implied the deletion of some other table columns, which had a great impact on the system code and required great attention to detail. Other times the functionalities were built from the ground up, so there was no need to alter what was already built.

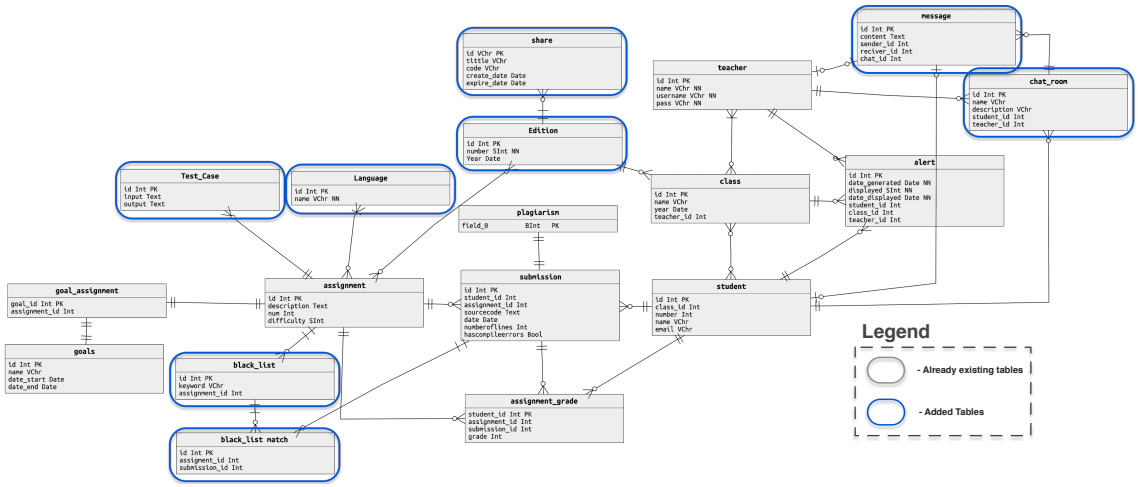


Figure 5.2: New Database Schema

The tables that have been added to the system's database are listed below, along with a brief explanation of why they have been included. Figure 5.2 demonstrates how the system database has evolved, with the original tables colored in grey and the new tables colored in blue.

1. **Edition:** This table was added to allow the distinctions between editions of the same course. For the system to distinguish between assignments of different editions the new table has many to many relationships with the assignments table.

2. **Test Case:** CodeInsights only supported 4 test cases. We removed the input/output fields from the assignments and created a new table called test cases. Then by creating many to many relations with the assignments the database is now able to represent assignments with various test cases.

3. **Output:** This entity represents the submission output and its result (i.e., its comparison between the expected output and the resulting one). Since there are now multiple test cases, we have to apply the same relation to their outputs, meaning we also needed a new table.

4. **BlackList:** This table was created to provide the ability to have multiple blacklist keywords.

5. **BlackList Match:** This entity was designed mainly to differentiate blacklist matches that could exist. It also possesses a relation with the submission table to represent snapshots that contain prohibited keywords.

6. **Share:** This entity was added to store the information about the shared code rooms, which can be viewed by the teachers and students. Its main attributes are its access link and its expiring date.

7. **Chat Room:** This entity was designed mainly to differentiate chat rooms that could exist.

8. **Message:** Storing messages is essential to showing chat histories, therefore this entity is crucial to represent each message's content and for which chat room was it sent.

5.1.4 Chat System

Establishing direct communication between instructors and students is key in supporting the student's progression. For this purpose, CodeInsight's new version has been designed with a chatting system in mind to facilitate the exchange of help requests and feedback between teachers and their pupils. There are numerous approaches when constructing chatting systems, some more real-time (i.e., in normal conditions with an almost negligible delivery latency) than others. Therefore we'll look at a few of them in this section and pick one that fits best our context. But before describing each way to achieve the exchange of messages, we must understand how web applications were originally developed and one of their major limitations.

Web applications use the Hypertext Transfer Protocol (HTTP) following a client server model, meaning the web client is always the initiator of the transactions (i.e., the client requests a resource via HTTP and then gets a response from the server, always following this order). This uni-directional attribute has been an HTTP limitation for quite some time since servers can not simply send information without a request. Nowadays our needs have changed and 'persisting connections' are quite useful for providing a real-time experience to users on the web. Next, we will describe the three main methods for replicating 'persisting connections': AJAX polling, AJAX Long polling, and WebSockets.

AJAX polling

One way of approaching this task would be to resort to the server's file system and store each chat as a file. Then using a looping function every deliberate amount of seconds re-read that file and show its content to the users. A simple modification to this approach would be to use the database of CodeInsights to store chat-related data instead of using files. This approach is known as AJAX (Asynchronous JavaScript and XML) polling and its basic concept is that the client requests data from the server on a regular basis. Figure 5.3 shows the basic flow of messages exchanged between clients and server with the standard AJAX polling technique.

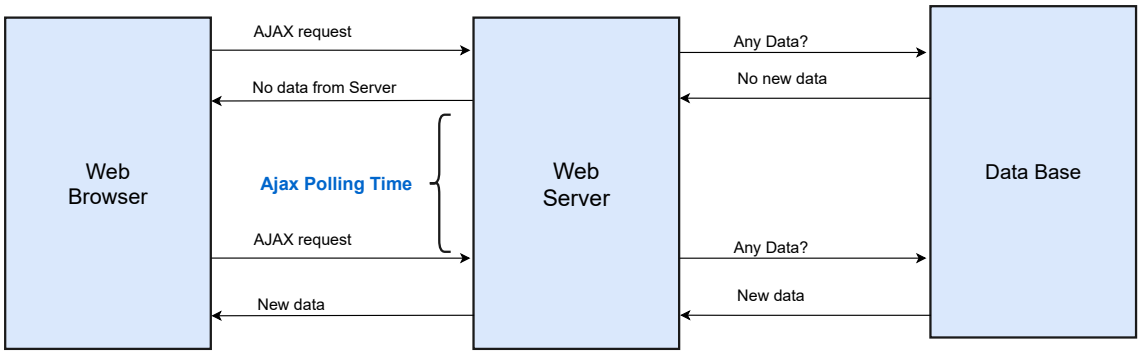


Figure 5.3: Ajax polling

AJAX long polling
Another way of establishing a persistent connection between servers and clients is long polling. This variation of the traditional polling technique allows the server to push information to a client whenever the data is available. With Long-Polling, the client requests information from the server exactly as in regular polling, but with the expectation that the server may not respond immediately. The server does not close the connection until it has a message to send. When a message appears the server responds to the request with it. By having this extra logic and only sending data when it is available, the number of HTTP requests is quite smaller in comparison with the less refined traditional polling.

WebSockets
The most common way of implementing these direct connections nowadays is by using WebSockets. A WebSocket is a bi-directional, full-duplex, persistent connection from a web browser to a server. Once a WebSocket connection is established the connection stays open until the client or server decides to close it. Clients connect to the server and proceed to send messages, while other clients listen to this connection and read their messages. The WebSocket protocol enables the communication between a client and a server with lower overheads, facilitating real-time data transfer from and to the server.

Trade-off Analysis
The final decision came up to the selection between the need for low latency and the complexity of implementing each technique. We ended up choosing the normal AJAX polling as it was a straightforward implementation and didn't require learning a whole new framework. This decision meant that clients would experience a bigger latency between sending and receiving messages. The latency would be defined by the polling interval that we decided would be thirty seconds. So no matter what, in the worst-case scenario a teacher or student would receive a message with a thirty seconds delay. This was within the metric value defined for our performance quality attribute detailed in section 4.2.

5.1.5 Share

Using the opposite reasoning used in the previous section 5.1.4, we opted to use WebSockets to construct the share system. While we did not consider latency to be a key factor in the chatting system, we did so here. The primary goal of sharing code is for students to observe and learn from code changes performed by the teacher in a class environment (i.e., in class while the teacher is talking over and explaining his changes). We ensured that users never experience immediate changes to blocks of code and thus losing information, by allowing pupils to view single character modifications. As a result of the latency demands, also depicted in section 4.2 , we chose WebSockets since the communication of

code between users would need to be almost instantaneous. In the following paragraphs, we describe our choices for the WebSocket Server and the Message Exchange Format.

WebSocket Server
WebSockets are based on HTTP, most of the frameworks available to implement WebSockets depend on an HTTP server. So another decision made was should be the framework used to implement a WebSocket server while taking into account our back-end language. One of the most common ways to implement a WebSocket server is to use the socketIO toll in conjunction with a nodeJs server. This approach would require running another HTTP server. Another alternative, and the one chosen is to use Ratchet, a PHP WebSocket library. Since it uses PHP, it would not require another server, and its integration with CodeInsight's backbone would be easier and simpler.

Message Exchange Format
A Websocket by itself quickly provides the foundation of communication between two or more browsers. But there is a need to add structure and format to the messages traded between the users and the WebSocket server. Each share room is generated by either students or teachers, and it is uniquely identified and accessed via a *link*. Since multiple students could be watching or even performing changes to the code, each room supports multiple connections from different users. Figure 5.4 represents the sequence of messages chosen for two different users to successfully trade code changes.

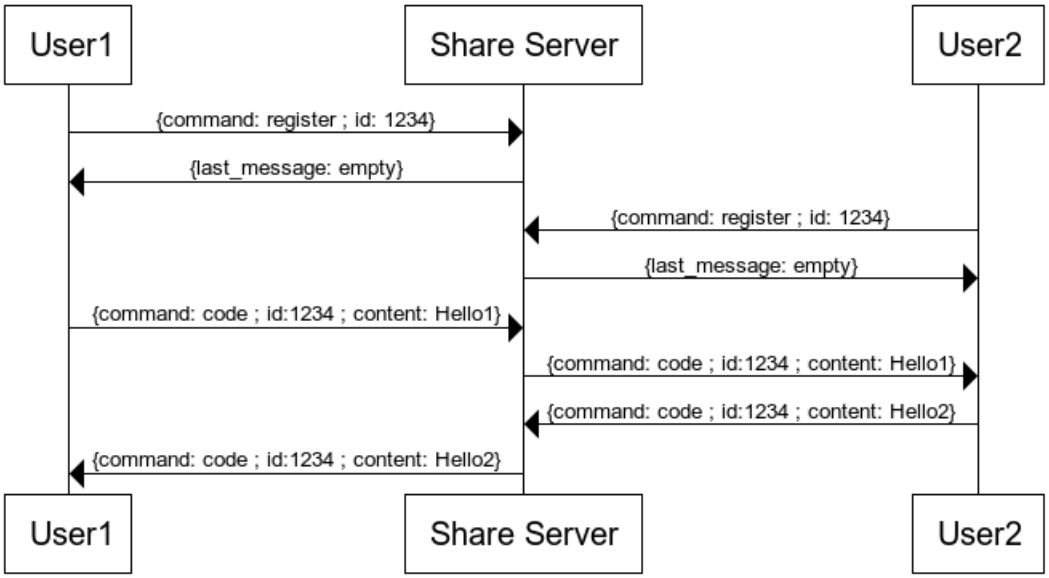


Figure 5.4: Share room, exchange of messages

As we can see each message is formatted in JSON and can be classified in one of two types: *Code* or *Register*. A *Register* message is essentially a request to join a chat room, this type of message must always include the chat room identifier. A *Code* message represents a change in the source code and it contains the new source code to be distributed across all users that are a part of the chat room.

5.1.6 Security

The sandbox server is in charge of running and validating the snapshots' code. As noted by the chromium project [41] a sandbox server is capable of creating processes that run in a highly restrictive environment. By restraining the processes to have access to only a part

of the system we greatly improve the security of the server. For example, it's fairly usual to limit the process's access to the machine's file system and its network. This process of limiting access is normally called jailing, and there are a lot of ways to achieve a jailed sandbox.

However, until this point the student's code was unrestricted and could perform any kind of operation on the host machine (i.e., the same machine that most of the time also hosted the web server for the CodeInsights). Therefore one of the goals for this project was to improve the security of the sandbox server and provide mechanisms to mitigate possible attacks.

During preliminary research, we concluded that to make a jailed sandbox we could not utilize a 'one does it all method'. As can be seen in figure 5.5 there are many APIs used by the chromium project to provide the safe execution of unknown code in a Linux environment. By using multiple tiers, security is maximized. For instance, even if for some reason the filesystem access is compromised there are other lines of defense in place.

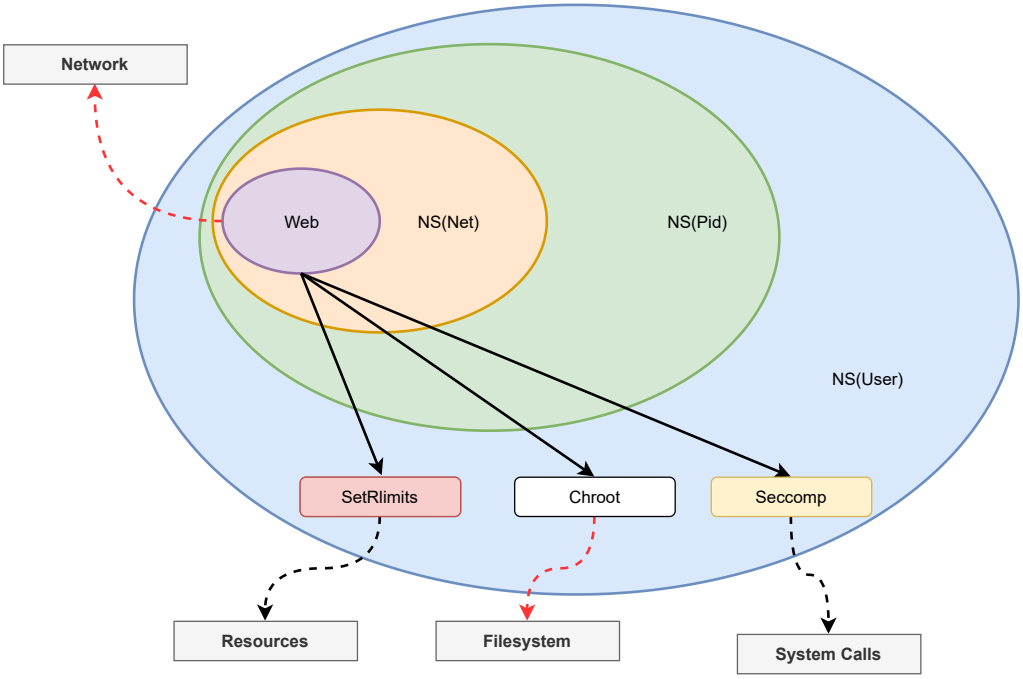


Figure 5.5: Chromium project - Different sandboxing techniques on Linux

There exist a lot of core Linux-related APIs like the network namespace that impedes access to the internet. Since we do not possess a high experience with this operating system we decided to use its higher level APIs to provide better security for CodeInsight's sandbox. The next list explains each API method and the techniques they use.

- **CHROOT:** A chroot jail is a way to isolate a process and its children from the rest of the system [29]. It should only be used for processes that don't run as root, as root users can break out of the jail very easily. The basic idea is to create a directory tree where only a copy of the system files needed for a process to run exist. Then by using the *chroot()* system call, we change the root directory to be at the base of this new tree and start the process running in that chroot'd environment. Since it can't reference paths outside the modified root, it can't perform operations (read/write etc.) maliciously on those locations. In essence, we limit access to the file system.

- **SEECOMP:** To restrict the actions and resources available to the student code, the untrusted user is limited to a whitelist of allowed system calls using the Linux seccomp library [30]. A system call is a way for applications to ask the OS kernel to do specific hardware and networking tasks on their behalf, since they alone don't have direct access to lower-level hardware. The system calls blocked should follow a white list approach for maximum security, since there are a large variety of system calls that might achieve the same goal. By using a blacklist approach we might miss some calls and only one is enough for the whole system to be compromised.

- **RLimits:** Rlimits, short for Linux memory limits, is a mechanism to constrain the system resources available for a process [25]. Limits can be imposed on a set of resources, such as memory, GPU, number of file descriptors open, and many more. With this tool every time a process attempts to violate one of these limits, a related signal is sent to the process (such as SIGXCPU for using too much of the CPU), which is then translated internally to a SIGKILL on execution of the student's code.

5.1.7 Client-side

The previous sections described some of the main changes applied to the main server and its inherent new components to provide the new system functionalities. This section focuses solely on the client side and describes some of the frameworks used to archive other features related to the users' display, such as dashboard customization.

Dashboard customization

The ability to personalize website pages to their users' tastes is not new, and a lot of frameworks have been developed to facilitate this process. We decomposed CodeInsights dashboard customization into three major operations: addition of graphics, deletion of graphics, changing the position of graphics and finally resizing graphics. The operation of adding and deleting graphics can easily be done by adding or removing DOM nodes (Document Object Model defines the standard for accessing and manipulating HTML and XML documents). Thus simple JavaScript operations were enough. Figure 5.6 shows the intended way of removing a graphic from the teacher's main dashboard.

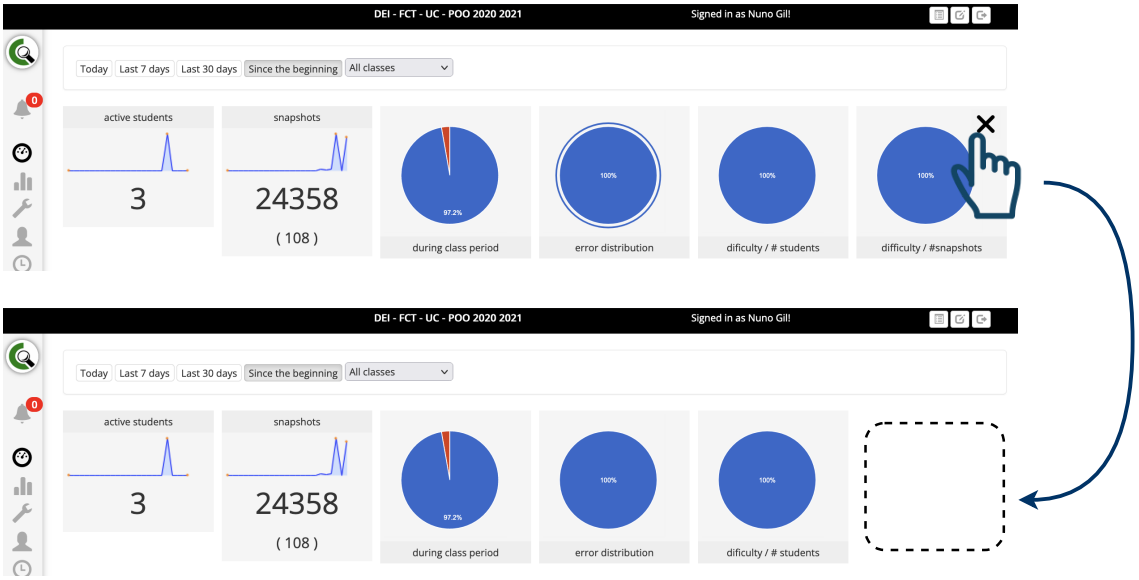


Figure 5.6: Removing a dashboard visualisation

On the other hand, moving graphics and changing their sizes is a lot harder, as it involves implementing methods to detect collisions. While choosing the correct frameworks for each operation some considerations were taken into account. The main ones were comparing each framework's features and the technologies it supported. In addition, the popularity of the framework was also taken into account as it reflects the use and consequent support the tool is given.

- **Jquery Ui Sortable:** This plugin makes selected DOM elements sortable by dragging with the mouse [49].
- **Jquery Ui Resize:** This plugin enables any DOM element to be resizable. With the cursor, a user can grab the right or bottom border and drag a DOM element to the desired width or height [48].
- **Mansory Layout:** Masonry is a JavaScript grid layout library. It works by placing elements in optimal position based on available vertical space, sort of like a mason fitting stones in a wall [12].

Rich text

Rich text editors or sometimes referenced as WYSIWYG (What You See Is What You Get) are locations in a web page that can be used to add, edit and format text. In CodeInsights they provide a way to better describe assignments to students. While looking at all the options available we took into account the popularity of the frameworks and their customization possibilities. We ended up selecting the TinyMCE rich text editor as it is the most popular tool of its kind and offers high customization via plugins [46]. A very active community and comprehensive documentation meant we would not have much difficulty getting started with TinyMCE. Additionally, its use of plugins provided an easy way of implementing the uploading and storing of images in the server database.

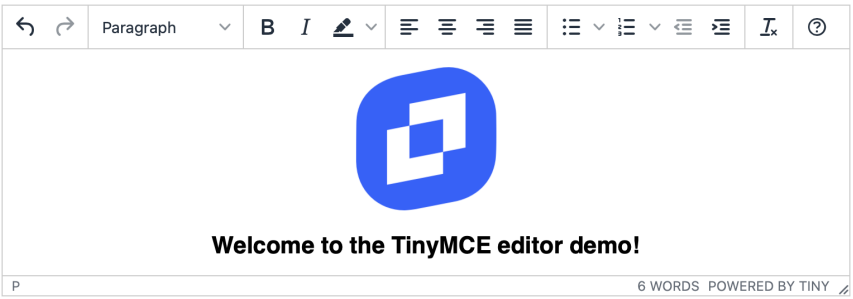


Figure 5.7: WYSIWY - TinyMCE default text editor

The TinyMCE default toolbar with all its functionalities is shown in Figure 5.7. We can see the standard text-styling options, such as the ability to underline specific sentences. In addition an image is already shown in the rich text editor.

Tags

Two CodeInsights assignment properties can be categorized and therefore could be seen as tags. These are the blacklisted keywords and the assignments topics. Instead of asking the teacher to provide a list of words separated by commas, we can ask them instead to enter tags one by one. By using tags like the ones in *StackOverflow* [45] we allow the user to better visualize and delete them. Figure 5.9 depicts how tags are usually displayed in web applications.

Figure 5.8: *StacksOverflow* [45] Tags

The library of choice was the Bootstrap-tagsinput [26], a simple plugin to the already in use Bootstrap grid system. Our decision was restricted by the Bootstrap version used by the previous system. CodeInsights uses version 3 of Bootstrap and in version 5 there is no need for a plugin, but we were constrained to the original version.

Notification Toast

Since we now had designed the system with a chatting system, one of the main requirements that this addition brought was the need for new messages notifications. Web sites usually resort to toasts, which are basically alert messages, to push information to their users.

Figure 5.9: Toasts from a generic web site

In order to provide a friendly way of users seeing and interacting with the arrival of messages, we choose a simple Javascript library called Toastr. Toastr provides a simple way of creating toasts and to handle click actions. This was pretty useful when redirecting the user to the chat of the message he just received.

5.2 GUI and UX Design

User interface (GUI) and user experience (UX) design are important parts of program development. Because users never see what's beneath the GUI, what they see is a complete reflection of what the application can do as far as they're aware. The user interface must tell users all they need to know about the application and assist them in fully utilizing its capabilities. Many of the design choices for the GUI made in the new version of CodeInsights were based on the prior CodeInsights application and a basic understanding of Ben Schneiderman's eight golden guidelines. Schneiderman [5] outlined the following 'Eight Golden Rules':

- Strive for consistency.
- Enable frequent users to use shortcuts.
- Offer informative feedback.
- Design dialogs to yield closure.
- Offer simple error handling.
- Permit easy reversal of actions.
- Support internal locus of control.
- Reduce short-term memory load.

Some components of the CodeInsights application GUI depict some rules better than others, however the following subsections show how the system follows most of these principles.

5.2.1 Strive for consistency

The previous version of CodeInsights was utilized by a significant number of teachers and students who were already familiar with the system layouts. So we tried to maintain its minimalistic web application look, maintaining the original color scheme and its characteristic left navigation bar. We attempted to make both applications as homogeneous as possible. For example, to support the entity management requirement we only added a button to the navigation bar.

5.2.2 Enable frequent users to use shortcuts

Applications should provide ways for experienced users to increase their efficacy. While designing and also through the development process we tried to create shortcuts, enabling proficient users to achieve the same operations with less click. For instance, a teacher can change the assignment optionality on the edit assignment page. However, if he is on the show all assignments page he can do the same by directly clicking the optional attribute square. These operations are identical but one can be done with three fewer clicks.

5.2.3 Offer informative feedback

Typically when designing an application, developers should have in mind that users should receive feedback on how their actions affected the system. Generally, when everything goes right he sees directly the alteration made to the system. However, it's just as crucial to let the user know when something is not working properly. For example, when adding or doing any operation related to entities, if, for some reason the database fails, the system (CodeInsights) indicates that the operation was not successful.

5.2.4 Design Dialogs to Yield Closure

This point relates to the building of a user journey and the designation of a start, middle, and finish point for each use case. The CodeInsights application achieves this propriety by using the same structures for similar use cases. For example, adding a student or a class always starts with the visualization of the overall entities and then the selection of the option to add a new one. Additionally by using a breadcrumb the user always knows in which part of the system he is at, and can feel confident navigating the application.

5.2.5 Offer Simple Error Handling

A good interface should always impede user errors by being as informative as possible, however, errors will happen so handling them is equally important. In CodeInsights most of its forms have different types (i.e., some are referent to emails, other to a number and so on), so if a user makes the mistake of typing the wrong data an error message is displayed over the form.

5.2.6 Permit Easy Reversal of Actions

As much as possible, actions performed by the user should not be permanent as they cause a sense of anxiety. By making easy reversal options, users are encouraged to explore as much as possible. In CodeInsights if a teacher makes a mistake of typing the wrong information while creating an assignment, he can always edit it later. This same principle applies to most of the other entities present in CodeInsights.

5.3 Mockups

A specific mockup was created for each functional requirement at the designing phase. All of CodeInsight's mockups were built using the Balsamiq website.

This section contains a mock-up of the UI that was created in the early stages of development. The mock-up in Figure 5.10 looks very similar to the final product, however, some mockups did not follow this same 'behaviour'. In general major mockups alterations were kept to a minimum.

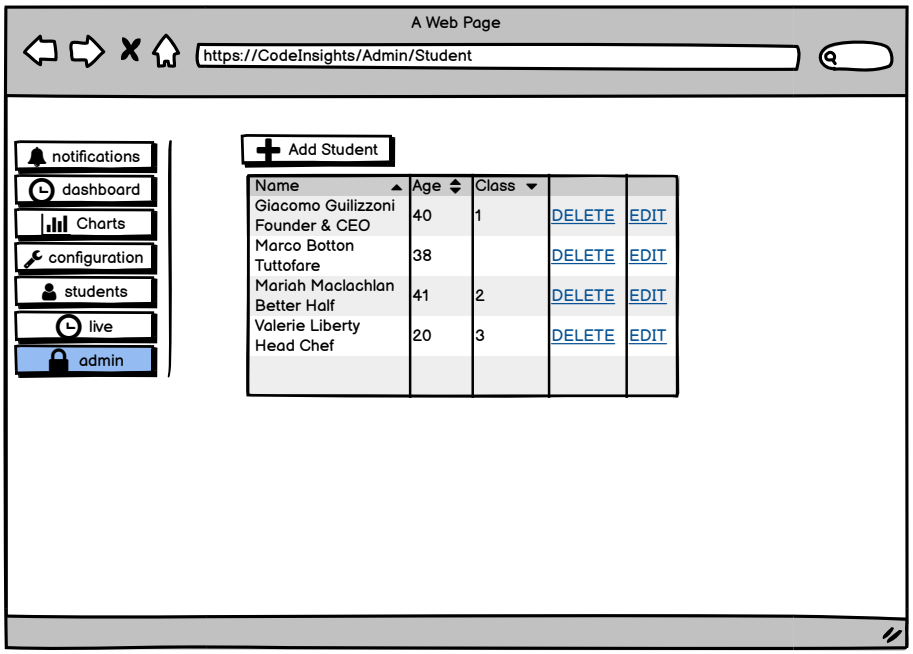


Figure 5.10: Edit student mockup

5.4 Putting it all together

Until now, we've primarily described our design decisions to meet the system's functional requirements at a component level. As a result, this chapter included a detailed description of how each new component would work and interact with the others. In addition, certain previously installed components were modified to achieve their new correct behavior. We can now zoom out to see all of the system alterations in a larger picture. Figure 5.11 depicts CodeInsight's architecture, including all of its components and how they interact.

A rectangle represents each component in the diagram. When a component is grayed out, it signifies it has largely remained unchanged. If a component is orange, it indicates substantial adjustments were required to change its behavior. Finally, if a component is colored blue, it implies that it was built from the ground up. The messages and data transferred between components are also color coded to highlight their interactions.

Even though we didn't go into detail about how the import/export data would operate due to its simplicity (i.e., there was no design choice), we can see simple Json structured messages used to transport information across systems in the schema. Another essential consideration when analyzing this figure is that some components can be broken into multiple pieces. For example, we introduced edition and entity management to the monitoring tool component.

Additionally, we can see all the components that were mentioned in the earlier parts of this chapter: the new WebSocket server, the forgot password mechanism and the chat system. Finally, the sandbox server and processing had to be changed to allocate the changes to the processing support and the multiple test cases.

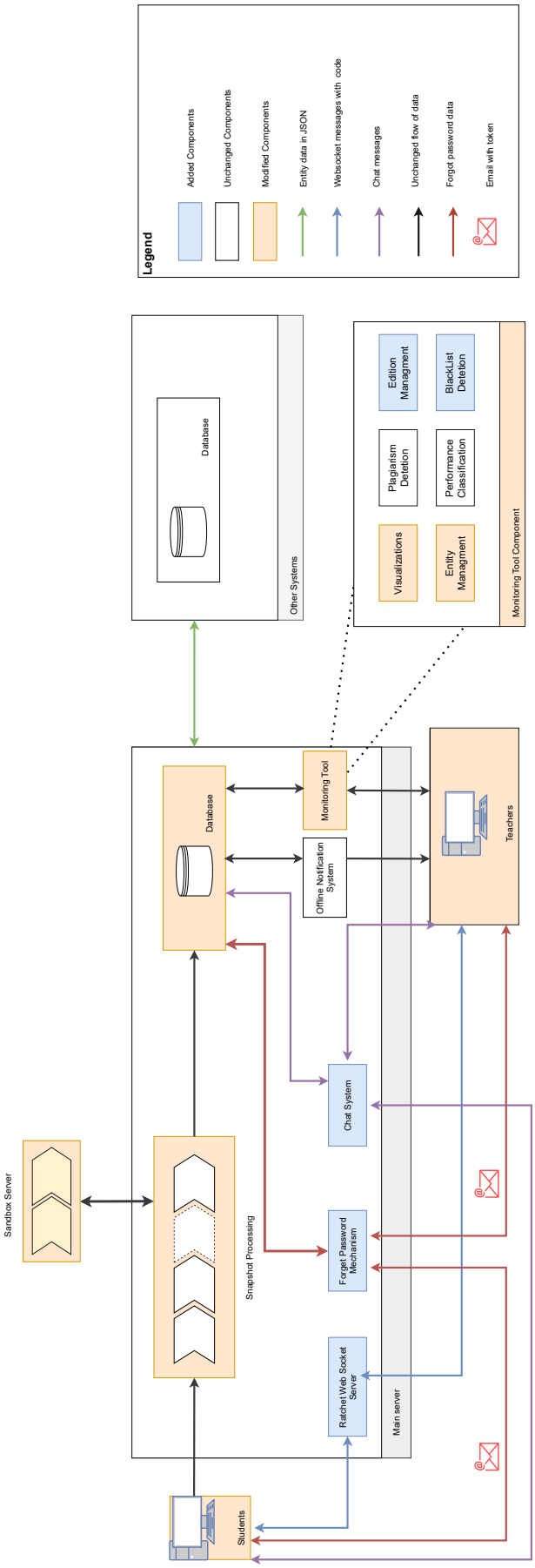


Figure 5.11: New architecture of CodeInsights

Chapter 6

Implementation

Taking into account the design choices covered in chapter 5, the following chapter will go over the implementation process of the new system functionalities. It starts in section 6.1 with an overview of the requirements that were planned and indicates which ones were successful and which ones were not successfully implemented. Every software project encounters challenges, which must be overcome quickly to keep the development process on schedule. Dealing with the new CodeInsight's edition management and the dependencies that its single edition structure had, was the most difficult component of the project. As a result, this chapter is also responsible for explaining some of the significant issues that were encountered in section 6.2. In the second half of the chapter, some of the final results of the implementation process are discussed in a rapid review of the system in section 6.3.

6.1 Implemented Requirements

The initial requirements planned and explained in chapter 4 were not all implemented. There are a variety of reasons that made some functionalities left to be implemented or partially done. Firstly due to the dynamic nature of CodeInsights some requirements were not initially planned, their addition consequently made other functionalities go down in the priority list. Additionally, some requirements that at first seemed simple turned out to be more complex and their implementation was just not worth the extra time it would take to add these features to the system.

Below, table 6.1 indicates the development status of each functional requirement, which can be categorized as fully, partially, or not implemented.

The majority of the important requirements (i.e., those with the highest priority) were either fully or partially implemented. The only exception is the dashboard customization

Category	ID	Requirement	Status
Entities	1	Managing students in the system	Finished
	2	Managing teachers in the system	Finished
	3	Managing classes in the system	Finished
Edition	4	Edition managment	Finished
	5	Edition selection	Finished
Passwords	6	Forgot password	Finished
Assignment	7	Importing exercises (Different editions)	Finished
	8	Inserting assigments (Test cases)	Finished
	9	Rich text description	Finished
System Integration	10	Data import	Finished
	11	Data export	Finished
Dashboard Customization	12	Deleting dashboard graphics	Finished
	13	Adding graphics to the dashboard	Finished
	14	Moving graphics on the dashboard	Unfinished
	15	Resizing graphics to dashboard	Unfinished

Category	ID	Requirement	Status
Processing	16	Accept processing code	Partially
	17	Display processing code	Partially
Chat	21	Starting a chat	Finished
	22	Sending a message via chat	Finished
	23	Chat Selection	Finished
Share	18	Create a code sharing room	Finished
	19	Share code	Finished
	20	View all of the share rooms	Finished
Security	24	Protection against malicious code execution	Partially
	25	Adding black list keywords	Finished
	26	Detecting black list keywords	Finished
Misc	27	Code peer-reviewing	Unfinished

Table 6.1: Functional requirements - Development status

where we faced difficulties that will be explained in 6.2.2. The other functional requirements left to be implemented were low on the priority list and hence were described as nice-to-have features. Also, we must note that some of the quality attributes, such as the performance of the sandbox server, were sophisticated enough that we now believe they would require a project of their own. For this reason, we did cover it on this project as its analysis was too superficial.

6.2 Interesting parts of implementation

Several challenges arose throughout the creation of the CodeInsights version, some of which were due to the steep learning curve required to master new frameworks, while others were far more complex. In the previous section 6.1 we detailed the status of each functionality's development where some functionalities were not fully implemented. As a result, the subsections that follow describe why certain capabilities were only partially realized and not fully developed.

6.2.1 Edition

In order to offer numerous editions per course to CodeInsights, we constructed a new table that identified which edition the teachers were presently seeing (i.e., the selected edition). Each assignment and class would now need to have an attribute that indicates which edition they belonged to. Students and submissions were tied to classes and assignments directly, and hence to an edition indirectly. As a result, any operation involving any of these entities now had to take into account the current edition chosen by a teacher. Consequently, the bulk of previous database accesses had to be filtered based on a certain edition.

One problem that surged mid-development was the system's anonymous submissions. CodeInsights uses annotations in source code to identify each one of the students submitting code, but if by some reason annotations are wrong the system attributes the submission to a dummy entity to not lose information. Since we now had several editions, we needed to add an anonymous student and assignment for each edition. This was done by adding a flag that indicated if the student or assignment was classified as dummy. Now when submitting wrongly identified code the system always checks the dummy student and dummy assignment for a particular edition.

6.2.2 Dashboard Customization

When designing the dashboard customization functionality, we choose several distinct libraries to achieve the resizing and shifting of graphics. Separately, each one of the libraries achieved with ease their main purpose, whether that would be moving or resizing a graphic. The issue started when we integrated all of them. When moving a resized graph it would present a flickering behavior, that is moving irregularly and unsteadily. The layout was affected by the constant irregularity in graphic movement, with a simple graphic movement being enough to cause all the others to alter their position. This erratic behavior made it hard to assure that teachers were in control of their actions.

Another issue faced was the persistence of each user layout. The storable plugin had a built-in option where it directly saved the order of the graphs in user local storage whenever an element was moved. The problem came when a user would add or remove graphics resulting in the plugin not storing these alterations.

We ultimately decided to only implement the adding and deleting of graphics because we had already invested a significant amount of work on the shifting and resizing procedures and no solution was in sight. This is the primary reason for the unfinished categorization is some of the Dashboard customization use cases.

6.2.3 Processing

The translation procedure began with the use of a large number of simple regular expressions. In Java, for example, when declaring a variable, we must specify the type of data it represents, such as an int. Variables in JavaScript do not have a type, instead of int, we have to use the let keyword instead (or the older syntax var). As a result, one of the regular expressions was responsible for finding all of the data types and replacing them with the var keyword.

The next step was to use a JavaScript AST to manipulate the student's code in order to move global variable assignments to the setup function. The first stage was getting to know the tool in hands, and learning about its basic functionalities of traversing and altering tree nodes. The library parses JavaScript code, according to ECMAScript specification, and generates an abstract syntax tree following the ESTree standard. So the next phase was to examine each tree node in the ESTree standard to determine which ones represented variable assignments and which ones represented function declarations.

Basically, we needed to move tree nodes with type Variable Declarators that were children of the Program node (i.e., the root of the tree and therefore a global variable declaration). In the end, the resulting algorithm followed the next pseudo-code:

Algorithm 1 AST processing algorithm

```
T (Sintax tree node)
G ← [] (Array of global variables)
E ← [] (Array of global variables expressions)
{Iterate trough the syntax tree}
while Iterate trough the T syntax tree do
    if T node type is "Program" then
        {Iterate trough the T sub syntax tree}
        while T is not null do
            {Find Global variable declarations and assignments}
            if T node type is "VaraibleDeclaratior" and contains and assignment then
                add the T node to the G array
            end if
        end while
    end if
    {Save setup funtion node for later}
    if T node type is "Function" and name is setup then
        S ← T
    end if
end while
{Add the variable assignment to the setup function}
for all g in G do
    add g(variable assignment) expression to S
end for
```

Once we had the basic algorithm implemented, we started testing it by attempting to translate Processing code to p5.js. The Processing examples were taken from the organization web-site [39] and applied to different kinds of concepts. Arrays, Camera, Control, Color, and Data were some of the coding concepts present in examples that were translated. When testing, some issues with the translation process were discovered such as the casting of variables. This problem was created because all variable types were being translated into var keywords, so in the code we would get *(var)variable_name*, which is not syntactically correct in JavaScript. In essence, we are basically translating Java to JavaScript. Due to the subtle variations and the large number of them, no language translation is ever faultless. We opted to mark the transition process as partially implemented, based on this very same logic.

6.2.4 Security

When starting to restrict the sandbox server processes' access we decided to start with the seecomp API. Soon after we realized that to restrain system calls we needed to modify the student's source code to embed our desired seccomp policy. Fortunately, there was an alternative, the so-called "zero code seccomp" [28]. Systemd is one of the popular implementations of a 'zero code seccomp' approach. Essentially it is a service manager that provides utilities to help with common system administration tasks.

The sandbox server works with Apache to receive HTTP requests and pass them to the back end where the code will be executed. PHP is responsible for executing the commands necessary for running the code, which sometimes implies compiling and then running the code. So the natural choice was to alter the Apache service to follow a set of rules. The final configuration file for the Apache service is shown in figure 6.1.

```
/etc/systemd/system/apache2.service.d/.#override.conf0078f61afdb64549
[Service]
PrivateDevices=true
ProtectControlGroups=true
ProtectHome=true
ProtectKernelTunables=true
ProtectSystem=full
RestrictSUIDSGID=true
```

Figure 6.1: Apache service configuration file

Some of the main properties that we can take from the of the service file are:

- **ProtectKernelModules** : Blocks explicit loading and unloading of kernel modules, drivers, etc. Particularly useful for Embedded Systems to make sure that your kernel is not modified.
- **ProtectSystem** : Makes the whole file system read only for the processes of the unit.
- **ProtectKernelTunables** : Prevents tuning and configuring some Linux kernel parameters during run-time.
- **ProtectControlGroups** : Prevents tuning resources parameters of processes.

As can be seen in the systemd manual [31], there are a lot of possible configurations. In the end, we did not have time to further explore this theme. For example, the PrivateNetwork parameter can block the process's access to the network, but we could not test the system

running with this configuration. However we still managed to block access to certain critical folders such as */home* (done via the ProtectSystem flag), but there can be a lot more done here. Fundamentally, this was a good starting point, and a possible future work can base itself on this approach to improve the sandbox server security.

6.3 CodeInsights Walkthrough

This section will act as a system's guide to performing the core functionalities of CodeInsights. Each subsection addresses a distinct type of functionality and displays it in the form of a screenshot. Our main goal is to show the results of some of the core features implemented during this project.

6.3.1 Teacher Home screen

The Home screen is the page that greats the teacher when logging in, it contains some of the main metrics captured by the tool. From the Home page, the teacher can access all the other system functionalities in the form of sections in a navigation bar. The most important alteration here was the addition of three new sections to the system: Message, Share, and Admin. In the top left corner, we can see the normal change password and log-out buttons. However, we added the change version button that redirects the teacher to the edition selection page, where he can change the edition he is currently seeing. Also, the name of the course and its edition is displayed in the top part of the Home page for identification purposes. Figure 6.5 shows the teacher home.

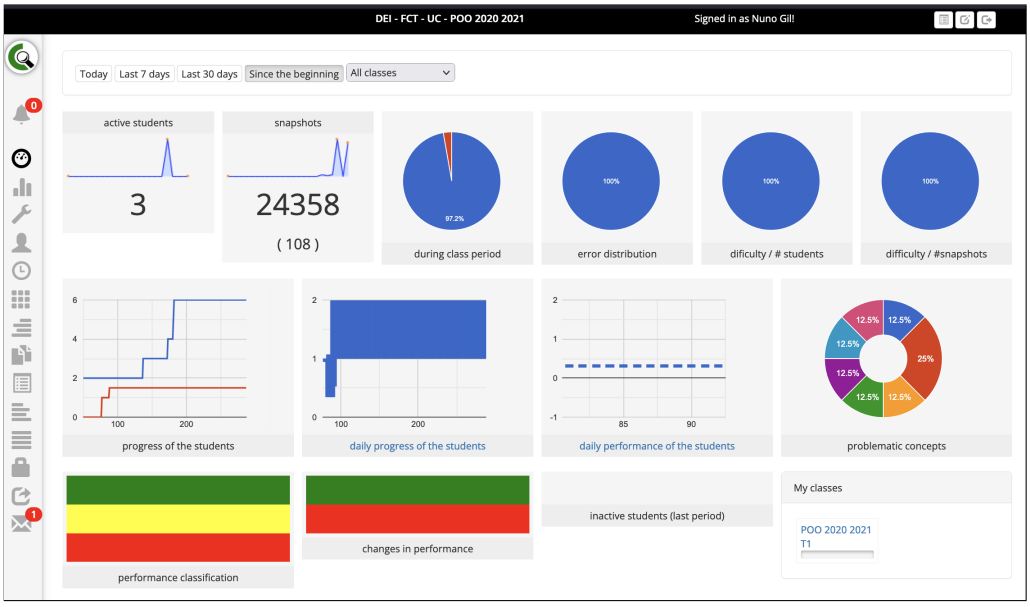


Figure 6.2: CodeInsight's Teacher Home Screen

6.3.2 Chatting

Teachers have access to all their chats through a dedicated Chat section. Since the student's view is not partitioned into sections like the teacher's, all their chats and help requests are shown directly on their home page. Chat lists are always ordered by the last message received date, meaning new messages will always appear in the first positions to the users. Additionally in the chat list, each user has their photo and name for identification purposes and the number of messages left unread. If a user clicks on a chat he is then redirected

to the submission page that originated the chat. From here both users can continue to exchange messages like normally. Figure 6.3 a chat taken from a submission page.

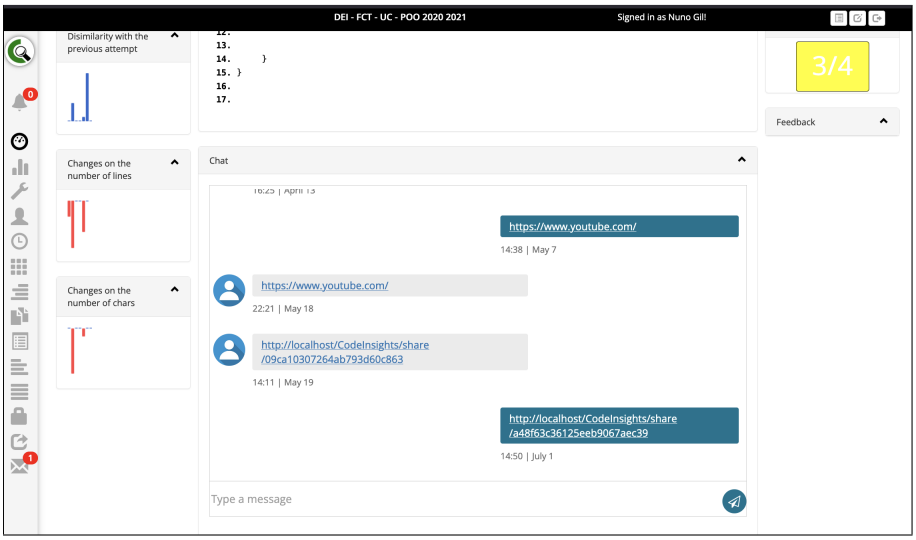


Figure 6.3: CodeInsight's Edition section - Import assignments

6.3.3 Share section

Finally, in the Share section, identified by a little arrow in the system navigation bar, we can see the available share rooms. In a similar faction, and once again aiming for consistency, a table with all the active share links is displayed. Each row represents a code sharing room and indicates some of its attributes like its creation and expiration date. From here it is possible to extend the duration of the code share room by another twenty-four hours or delete it completely from the system database. The teachers can then access the room by clicking the share room link. Once the teacher does this, he will be redirected to another page where he will be presented with a web code editor. In an identical manner to *CodeShare* [44] this page presents the possibility to download the code to a text file or to create a new code-sharing room.

Figure 6.4: CodeInsight's Share section - Web text editor

6.3.4 Admin section

The Admin section, represented by a lock, is where teachers can manage students, classes, courses, and editions. The navigation inside this section always follows the same structure. So first the teacher needs to indicate which kind of entities he would like to manage. After selecting the right entity type there is always a table displaying all of the entities

of that category. Each row represents a different entity and its attributes and all the manipulations possible that apply to that entity. For example deleting, updating, or even exporting it in a JSON format.

Specifically from the edition management, it is possible to import assignments from different editions. Figure 6.5 represents the page where this process is accomplished.

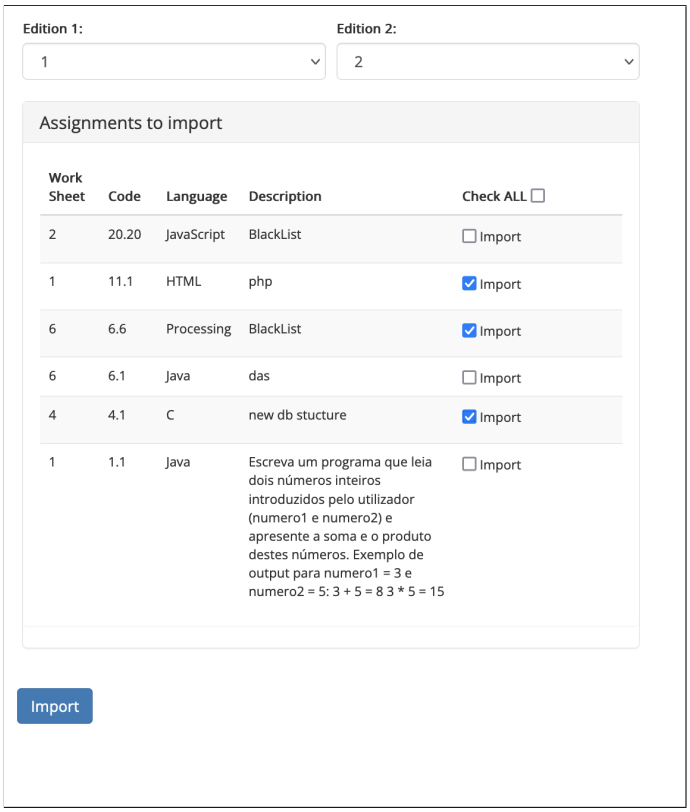


Figure 6.5: CodeInsight's Edition section - Import assignments

Chapter 7

Planning

The earlier chapters described how the whole process of bringing new functionalities to CodeInsights occurred, which took approximately twelve months. This chapter is responsible for documenting the project's planning, from its conception to development. In section 7.1 we look at what life cycle we chose and why it was selected. The roles that each individual involved in the project took are presented in section 7.2. The first semester planning, where the project conception took place, is depicted in section 7.3. In Section 7.4 we show the plan for the development phase (i.e. the second semester) and explain some of the reasons for its extension.

7.1 Life cycle

The choice of a project's life-cycle is one of the most, if not the most, important decisions to be made during software design. Different life-cycles have different stages and properties, making them appropriate for different kinds of problems and projects. Thus the resulting life-cycle of this choice should fit the project's purpose and not the other way around like it tends to occur. Generally, we can divide those life-cycles into two categories: plan-driven or agile

Plan-driven is characterized by its strong belief in predictability and high assurance. Waterfall, although not the only one, is a typical example of this type of approach. On the other hand, Agile methods do not fix all the plans at the beginning of a project. Instead, the project is broken into smaller sub-tasks that are implemented in short time-boxed iterations, to produce shippable code incrementally. But as Barry Boehm[4] states, it is a matter of balancing agility and discipline. So in the end, the chosen lifestyle was neither an agile nor plan-driven approach, and in fact we can not actually say it is a life cycle. Due to its informality we rather say our choice was to follow some of the values and attributes in each of the approaches.

The more formal and heavy documentation fits better my requirement engineering background, but the ability to incrementally provide a working product is highly valuable. Additionally, having the possibility of adapting to changing requirements is quite useful in an environment where the product is being used and constantly receiving feedback from users. Finally, the whole development of the project is by itself a challenge due to my inexperience with the tools used to develop CodeInsights. The highly agile approach of re-prioritizing requirements would ensure that no hang ups were created.
The processes used in the development of the new CodeInsights are depicted in Figure 7.1. But before exploring the overall development phase, we can not stress enough how this is by no means a formal approach and functions more as a set of general guidelines.

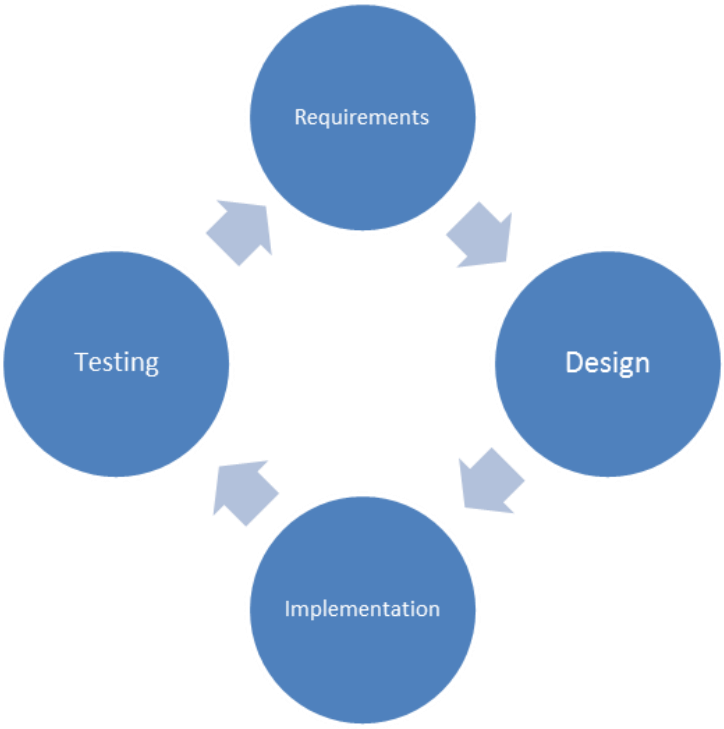


Figure 7.1: Development cycle throughout the development phase

Initially, a set of requirements were created in a plan-driven manner. A formal document, the requirements specification, was established and utilized as a product backlog to explain all of the system's new functionalities. The following stages were composed by the design and implementation of the requirements. The implementation followed a two weak iterations format, where a set of functionalities was selected by order of priority. Finally, at the end of the sprint-like iteration, a set of tests were performed to validate the system's improvements. The processes of Requirements, Design, Implementation, and Testing make up the basic cycle of the development phase. New features that were not originally anticipated were added to the prior bulk of requirements as the project progressed and the cycles repeated themselves.

7.2 Project Roles

Software projects are never easy and without the right planning and communication between all the individuals involved, they can become much harder. People or groups affected by a software development project are referred to as 'stakeholders'. Stakeholders can be found both inside and outside software development companies and their respective development teams. The stakeholder feedback informs the organization about the type of software required, as well as features and problems that must be addressed. Software projects can only move forward when the key stakeholders are all in place. Clear-cut roles and responsibilities within the team help the project to move forward and multiply chances of success.

Even though the CodeInsights upgrade project only involves a few people, each one of them plays a different yet vital role in the accomplishment of the project. Of all the roles we must highlight the influence of the project creator and contextualize it a bit more. In

short, the product creator formed a bridge between the project team and stakeholders. Nuno (the project creator), through the use of his program in the practical environment of his programming classes, had some concepts and ideas for the new improved version of the system. Additionally, feedback from users both teachers and students who used the program was also taken into account in the formulation of the new features of the system.

In essence, the original project creator mirrored the viewpoints of the system's end-users and their opinions on how the system could be improved. Based on this information, I, the software engineer, worked on both the front-end and back-end to materialize these concepts and ideas. Good communication was maintained throughout the project to minimize any potential differences in our points of view regarding some functionalities. Based on the methodology chosen to develop the project, every two weeks a new batch of new functionalities was implemented. Demonstrations, although in a not so formally declared interval, took place regularly validating or not the new features of the system. Normally the outcome of the demonstrations would be quite positive with minor alteration to the visual component of the new functionalities.

7.3 Schedule 1º Semester

In the first semester, the work was mainly preparation and planning for the second semester, where most of the development would take place.

It all began in September with a meeting between all of the 'team members' who would be responsible for improving CodeInsights. This get-together was held to provide context for the project and to establish some broad objectives. After that, my first objective was to read some of Nuno's scientific papers regarding CodeInsights, as I had no idea what tool I would be enhancing. In the meantime, I also looked into the monitoring tool ecosystem to learn more about each system's capabilities and limits. These chores took a little longer than anticipated, but nothing unusual.

Figure 7.2 depicts the first semester's planning, including all of its tasks. The estimated time to complete each task is shown in green. The effective (real) time for each one is shown in red.

Figure 7.2: First Semester Planning - Gantt Chart

We were now in early November, and with all of my newfound knowledge, I was ready to begin composing the system requisites. This task was expected to take two weeks, but it took four weeks instead. This delay can be easily explained by the formal way each use case was described and the vast number of requirements refined.

Finally, the first semester would conclude with the design of new functionalities (i.e. a high level architecture of our tool) and, if possible, some early implementation. Due to the prior two-week delay, this meant it would not be feasible to start the development phase, as I only had time to write the midterm report.

7.4 Schedule 2º Semester

We had a total of eight sprints in the second semester, which encompassed the whole new CodeInsight's development phase. The sprints were structured and prioritized based on the following criteria: implementation priority, projected implementation time, and task dependencies. Above all, tasks were restrained by what time available we had (i.e. the 'budget' for the new improved version of CodeInsight).

Figure 7.3 depicts the second semester development plan in a similar manner to the previous section, with the expected timeline colored in green and the actual timeline colored in red. As can be seen, there are various discrepancies between the actual vs expected results.

Figure 7.3: Second Semester Planning - Gantt Chart

The first task was to restructure the database, as this would affect the majority of the subsequent sprints. The entity and edition management were then implemented. We first underestimated the time it would take to address all of the edition management dependencies, which took around an extra week of work. During sprint 2, we must note the forgot password feature was also added to the list of features that needed to be implemented.

The next weeks went without a hitch, and we eventually managed to shorten our delay in the system integration requirement (sprint 4). By choosing simple JSON documents to transfer data between systems, we were able to complete this assignment in less time than we anticipated.

By now we were in April and about to start the dashboard customization with a 1-week delay. We realized that to keep the project on track some functionalities in sprint 7 had to be cut. After deciding to not implement the dashboard resize and move option we advanced to sprint 8, where we implemented the chatting system. In this sprint, yet again surged a priority requirement not initially planned, the sharing of code.

Due to technical difficulties (e.g. the dashboard customization plugin malfunctions) and the surge of new features (e.g. the share code mechanism) we arrived at the end of the semester behind schedule. However, this was perfectly fine since we were following the flexible guidelines of the Agile methodology. During this period, we considered and took the possibility of extending the project deadline, as we could use the extra time to polish existing features or add some that were missing.

7.5 Schedule Extension plan

Throughout the project's development some of the least prioritized requirements were not implemented, due to a variety of reasons already explained. With the opportunity to extend the project's deadline, we created a new plan for the upcoming months. Figure 7.4 represented the extended plan for the final months of this project. As usual, the expected time for each task is represented in green, while the actual time each one took is shown in red.

Figure 7.4: Extension Planning - Gantt Chart

The first task to be revisited was the dashboard customization, where we took another two weeks into trying to implement the missing features. Then we moved on to improving the processing translation so that it could support even more code.

By now we were in the middle of the plan and started addressing security of the system. The complexity in using systemd services to block the sandbox's access to the file system led to a week of delay. Finally, the rest of the time was reserved for general polishing and the writing of the final report.

Chapter 8

Conclusion

The purpose of this thesis was to improve CodeInsights, an existing programming monitoring tool. By adding several new features we made the teacher's job of overseeing their students' work even easier. The entire process of enhancing the application was presented, beginning with the motivation behind such improvements that led to the formation of this project, followed by an analysis of the tool in question and the monitoring tool ecosystem, and finally the formulation and implementation of these improvements.

In the first semester, we were able to define the functional aspects of the new versions of CodeInsights. To accomplish so, we needed to do a quick study of both the tool to be improved and the other systems in the monitoring programming environment. Combining this analysis with the ideas of the creator of the system, we managed to build a set of new requirements. Using the new features, we were able to design the new system architecture.

Also during this semester, a set of mockups were created to guide the development of each requirement. Since the system had previously been utilized and is currently being used, we had some usability guidelines in consideration, the most important being consistency. We tried matching the navigation and structure of the system, to make a future transaction between system versions as seamlessly as possible to users.

The development phase began in the second semester. During this phase, we followed several agile principles, which resulted in two-week sprint interactions. Around the end of the planned timeline the opportunity to extend the project deadline surged. We took this chance to mainly polish the already implemented features, and cover some of the features left out during the normal period.

Some low priority features were not implemented due to the addition of unplanned features or the underestimating of the complexity of some tasks. The continuation of the security subject would be a great starting point for the future work of this project. In addition, instead of requiring the user to be logged in to the system, a Json Web Token might be used to export entities. The ongoing use of CodeInsights will always require the development of additional functionalities to address new users' problems.

In summary, the application upgrade was successfully assessed, created and implemented, and this thesis detailed every step of the process. Despite a few noted flaws, the application is currently fully functional, and former users will soon be able to access it.

References

[1] Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005.

[2] H. Amer and W. Ibrahim. Using the ipad as a pedagogical tool to enhance the learning experince for novice programing students. In *2014 IEEE Global Engineering Education Conference (EDUCON)*, pages 178–183, 2014.

[3] H. Amer and W. Ibrahim. Using the ipad as a pedagogical tool to enhance the learning experince for novice programing students. In *2014 IEEE Global Engineering Education Conference (EDUCON)*, pages 178–183, 2014.

[4] B. Boehm and R. Turner. Balancing agility and discipline: evaluating and integrating agile and plan-driven methods. In *Proceedings. 26th International Conference on Software Engineering*, pages 718–719, 2004.

[5] Capian. Capian. shneiderman's eight golden rules of interface design. `https://capian.co/shneiderman-eight-golden-rules-interface-design`. Accessed: 2021-10-08.

[6] Tirupathi R. Chandrupatl. Quality and reliability in engineering. `https://assets.cambridge.org/97805215/15221/excerpt/9780521515221_excerpt.pdf`. Accessed: 2021-08-01.

[7] Tosti Hsu-Cheng Chiang. Analysis of learning behavior in a flipped programing classroom adopting problem-solving strategies. *Interactive Learning Environments*, 25(2):189–202, 2017.

[8] D. Chuda, P. Navrat, B. Kovacova, and P. Humay. The issue of (software) plagiarism: A student view. *IEEE Transactions on Education*, 55(1):22–28, 2012.

[9] Alistair Cockburn. 3] basic use case template. 01 1998.

[10] Codecademy. What is an ide? `https://www.codecademy.com/articles/what-is-an-ide`. Accessed: 2021-09-01.

[11] Fatima Abu Deeb and Timothy Hickey. The spinoza code tutor: Faculty poster abstract. *J. Comput. Sci. Coll.*, 30(6):154–155, June 2015.

[12] David DeSandro. Masonry cascading grid layout library. `https://masonry.desandro.com`. Accessed: 2021-06-01.

[13] Marcia Devlin and Kathleen Gray. In their own words: a qualitative study of the reasons australian university students plagiarize. *Higher Education Research & Development*, 26(2):181–198, 2007.

[14] Nicholas Diana, Michael Eagle, John Stamper, Shuchi Grover, Marie Bienkowski, and Satabdi Basu. An instructor dashboard for real-time analytics in interactive programming assignments. In *Proceedings of the Seventh International Learning Analytics amp; Knowledge Conference*, LAK '17, page 272–279, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Ryann K. Ellis. Efield guide to learning management, astd learning circuits. 2009.

[16] N. G. Fonseca, L. Macedo, M. J. Marcelino, and A. J. Mendes. Augmenting the teacher's perspective on programming student's performance via permanent monitoring. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2018.

[17] N. G. Fonseca, L. Macedo, and A. J. Mendes. Using early plagiarism detection in programming classes to address the student's difficulties. In *2018 International Symposium on Computers in Education (SIIE)*, pages 1–6, 2018.

[18] Ben Fry. Processing language organization home screen. `https://processing.org`. Accessed: 2021-05-01.

[19] Nuno Gil Fonseca, Luis Macedo, and Antonio Mendes. Monitoring the progress of programming students supported by a digital teaching assistant. pages 75–86, 08 2017.

[20] Nuno Gil Fonseca, Luís Macedo, and António José Mendes. Supporting differentiated instruction in programming courses through permanent progress monitoring. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 209–214, New York, NY, USA, 2018. Association for Computing Machinery.

[21] Stalgia Grigg. Processing transition. `https://github.com/processing/p5.js/wiki/Processing-transition`. Accessed: 2021-05-01.

[22] Red Hat. What is an ide? `https://www.redhat.com/en/topics/middleware/what-is-ide`. Accessed: 2021-09-01.

[23] IBM. Architectural characteristics of web-based applications. `https://www.ibm.com/docs/en/db2-for-zos/11?topic=environment-architectural-characteristics-web-based-applications`. Accessed: 2021-03-01.

[24] IBM. Dealing with constraints in software architecture design. `https://www.neverletdown.net/2014/10/dealing-with-constraints-in-software-architecture.html`. Accessed: 2021-05-01.

[25] IBM. setrlimit — control maximum resource consumption. `https://www.ibm.com/docs/en/zos/2.3.0?topic=functions-setrlimit-control-maximum-resource-consumption`. Accessed: 2021-08-01.

[26] Luckner Jr. Jean-Baptist. Bootstrap tags input. `https://bootstrap-tagsinput.github.io/bootstrap-tagsinput/examples/`. Accessed: 2021-06-01.

[27] Tony Jenkins. On the difficulty of learning to program. In *Loughborough University*.

[28] Ignat Korchagin. Sandboxing in linux with zero lines of code. https://blog.cloudflare.com/sandboxing-in-linux-with-zero-lines-of-code/. Accessed: 2021-08-01.

[29] Linux manual page. chroot - change root directory. https://man7.org/linux/man-pages/man2/chroot.2.html. Accessed: 2021-08-01.

[30] Linux manual page. seccomp - operate on secure computing state of the process. https://man7.org/linux/man-pages/man2/seccomp.2.html. Accessed: 2021-08-01.

[31] Systemd manual page. Execution environment configuration. https://www.freedesktop.org/software/systemd/man/systemd.exec.html. Accessed: 2021-08-01.

[32] Marco Marchiò. Peast (php ecmascript abstract syntax tree). https://github.com/mck89/peast. Accessed: 2021-05-01.

[33] John McKeogh and Chris Exton. Eclipse plug-in to monitor the programmer behaviour. pages 93–97, 01 2004.

[34] Christian Murphy, Gail Kaiser, Kristin Loveland, and Sahar Hasan. Retina: Helping students and instructors based on observed programming activities. *SIGCSE Bull.*, 41(1):178–182, March 2009.

[35] Cindy Norris, Frank Barry, James B. Fenwick Jr., Kathryn Reid, and Josh Rountree. Clockit: Collecting quantitative data on how beginning software developers really work. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, page 37–41, New York, NY, USA, 2008. Association for Computing Machinery.

[36] University of Maryland. Findbugs, a program which uses static analysis to look for bugs in java code. http://findbugs.sourceforge.net. Accessed: 2021-05-01.

[37] University of New Brunswick. Understanding quality attributes. https://www.cs.unb.ca/~wdu/cs6075w10/sa2.html. Accessed: 2021-05-01.

[38] University of Utah. Function definition. https://www.cs.utah.edu/~germain/PPS/Topics/functions.html. Accessed: 2021-09-01.

[39] Processing organization. Processing code examples. https://processing.org/examples/. Accessed: 2021-08-01.

[40] OWASP. Forgot password cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html. Accessed: 2021-05-01.

[41] The Chromium Projects. Sandbox faq. https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/design/sandbox_faq.md#What-is-the-sandbox. Accessed: 2021-05-01.

[42] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. Internet of things patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs*, EuroPlop '16, New York, NY, USA, 2016. Association for Computing Machinery.

[43] Amazon Web Services. What is a relational database? `https://aws.amazon.com/relational-database/`. Accessed: 2021-09-01.

[44] Code Share. Share code in real-time with developers. `https://codeshare.io`. Accessed: 2021-06-01.

[45] Stackoverflow. Stackoverflow website. `https://stackoverflow.com`. Accessed: 2021-10-01.

[46] Tiny Technologies. Tinymce the rich text editor behind great content creation experience. `https://www.tiny.cloud/tinymce/`. Accessed: 2021-06-01.

[47] Cassia Trojahn dos Santos. How to write a good state of the art: should it be the first step of your thesis ? In Maxime Lefrançois, MINES Saint-Étienne, and France, editors, *Rencontres des Jeunes Chercheurs en Intelligence Artificielle 2019*, Actes des Rencontres des Jeunes Chercheurs en Intelligence Artificielle 2019, Toulouse, France, July 2019. Co-localisées avec la Plate-Forme Intelligence Artificielle (PFIA 2019).

[48] Jquery UI. Resizable plugin. `https://jqueryui.com/sortable/`. Accessed: 2021-06-01.

[49] Jquery UI. Sortable plugin. `https://jqueryui.com/sortable/`. Accessed: 2021-06-01.

[50] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, page 117–122, New York, NY, USA, 2013. Association for Computing Machinery.

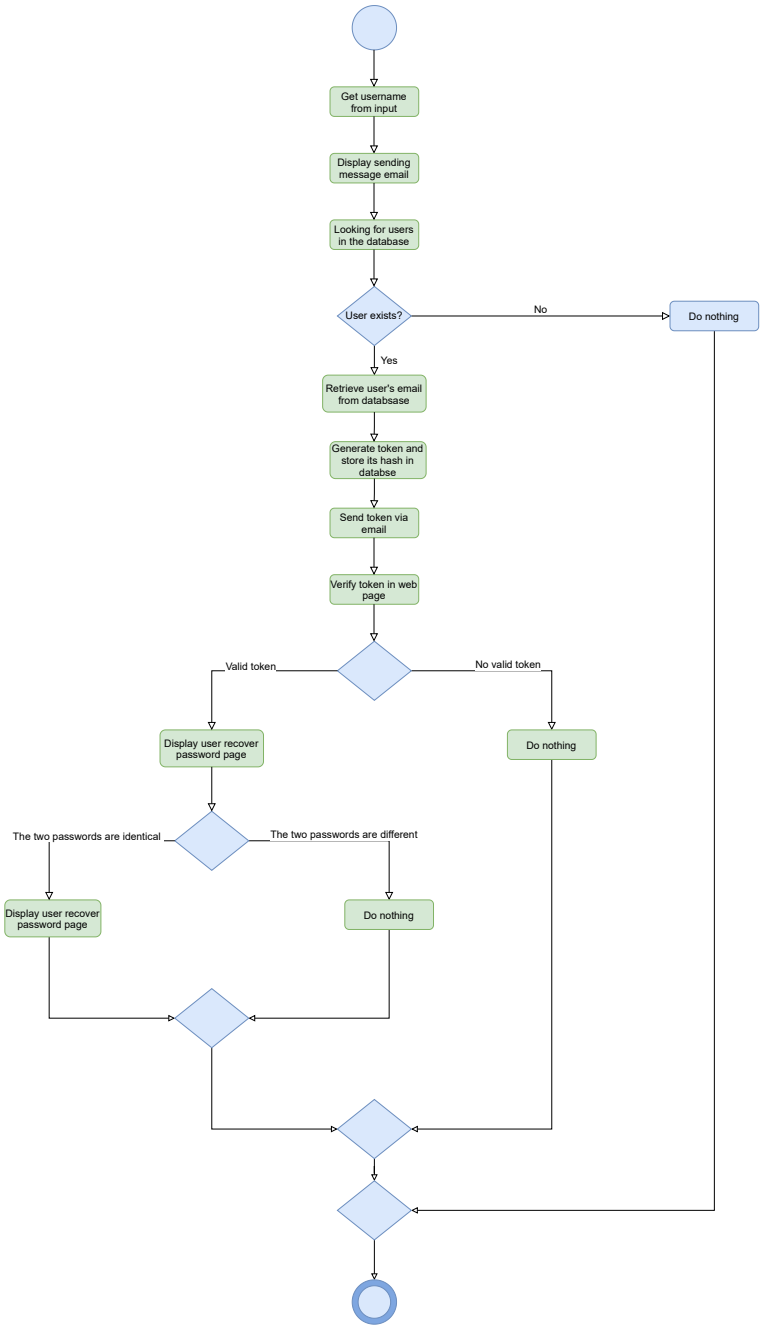
[51] Maximilian Rudolf Albrecht Wittmann, Matthew Bower, and Manolya Kavakli-Thorne. Using the score software package to analyse novice computer graphics programming. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, page 118–122, New York, NY, USA, 2011. Association for Computing Machinery.

[52] Qianqian Ye. P5.js organization home screen. `https://p5js.org`. Accessed: 2021-05-01.

Appendices

This page is intentionally left blank.

Appendix A


Get username from input
Display sending message email
Looking for users in the database
User exists?
No
Do nothing
Yes
Retrieve user's email from databsase
Generate token and store its hash in databse
Send token via email
Verify token in web page
Valid token
No valid token
Display user recover password page
Do nothing
The two passwords are identical
The two passwords are different
Display user recover password page
Do nothing

This page is intentionally left blank.

Appendix B

Functional Requirements - CodeInsights

Duarte André Teresa Guerreiro

Functional Requirements

Table 1 shows a broad view of all the functional requirements elicitation for the development phase of CI. In this section we will go over each high level requirement, explain its need and show some use cases. All this is done while following the priority given to each requirement.

Category	ID	Use Case	Priority
Entities	1	Managing students in the system	High
	2	Managing teachers in the system	High
	3	Managing classes in the system	High
Edition	4	Edition managment	High
	5	Edition selection	High
Passwords	6	Forgot password	High
Assignment	7	Importing exercises (Different editions)	High
	8	Inserting assigments (Test cases)	High
	9	Rich text description	High
System Integration	10	Data import	High
	11	Data export	High
Dashboard Customization	12	Deleting dashboard graphics	High
	13	Adding graphics to the dashboard	High
	14	Changing the layout of graphics on the dashboard	High
	15	Changing the size of graphics to dashboard	High
Processing	16	Accept processing code	Medium
	17	Display processing code	Medium
Chat	18	Starting a chat	Medium
	19	Sending a message via chat	Medium
	20	Chat Selection	Medium
Share	21	Create a code sharing room	Medium
	21	Share code	Medium
	23	View all of the share rooms	Medium
Segurança	24	Protection against malicious code execution (System)	Medium
	25	Adding keywords to a teacher's blacklist	Medium
	26	Protection against malicious code execution (Teacher)	Medium
Misc	27	Code peer-reviewing	Low

Table 1 – Functional Requirements - Use cases

Since there is a large number of use cases because each high level functional requirement was broken down into multiple finer detail ones, we decided to only show one per "Category". For example, Entity management is composed of the addition, deletion, and edition of various entities, but since they are also identical, we only cover one entity.

Edition Management

Instructors that teach programming do it on a year-to-year basis. For example, it's fairly uncommon for a teacher to teach java one year and then teach it the next with only a few minor modifications to its syllabus. In this report, a course edition refers to each academic year of a specific course.

The prior CodeInsights version was conceived with the assumption that each system would be independent from each other, working like local installations. Each course, as well as its several editions, would have its own CodeInsights installation. Installations would be accessed through various domains and have different databases. Generally, the domains would follow a template of /server_ip/CI_course_name_edition_number, and each course would be uniquely identified

by changing the course name and its edition. Because the system saves sensitive information such as student images, the independence across installations means there will be no shared information between various courses, which is advantageous in terms of security. Furthermore, dividing the system increased its performance by reducing the amount of data to be processed and metrics to be calculated.

In essence, each installation/edition would have identical databases and web servers, but they would be segregated. This meant that, if one instructor were to be using CodeInsights for an individual course, in the following edition of that course, he would have to request another installation from the administrator.

Since then, this premise has changed, and some instructors would like to have access to multiple editions of the same courses. Allowing teachers to effortlessly switch between editions would provide them access to past year's stats and visualizations, as well as information about previous assignments. Table 2 indicates the whole process in creating a new edition for a course.

Use case Name:	Add new edition to a course
Primary Actor:	Teacher
Pre-conditions:	The teacher must be in a valid session
Main Flow:	**1.** The teacher navigates to the course management page in the administration section. **2.** The teacher tells the system that he wants to add a new course. **3.** The system shows a form with all the information to be filled. **4.** The teacher enters all the information requested and submits the form. **5.** The system shows a message confirming the creation of the course in the system.
After conditions:	The system redirects the teacher to the course management page.
Alternative Scenario:	**4a.** Invalid information is entered by the administrator (Negative ages or email). **4a1.** Error information is displayed over the field in question. **4b.**The administrator wants to immediately associate the teacher with an edition of a course. **4b1.** The administrator of all existing courses can choose several.

Table 2 – Add a chart to the dashboard - Use Case

Test Cases

One of the many capabilities of CodeInsights is its ability to automatically grade source code via test cases. Initially, the systems supported up to two test cases per assignment for its output comparison. Over time a lot of teachers recommended the addition of more test cases, which would benefit the system as it would prevent hard-coded solutions. The previous version of CodeInsights had a limit of four test cases since the system was built from the ground up with a static approach to addressing test cases. As a result, the new version of the system was built with the prospect of having an unlimited number of test cases in mind. The actions involved in adding assignment test cases are represented in table 3.

Use case Name:	Insert assignments with multiple test cases
Primary Actor:	Teacher
Pre-conditions:	**1.**The teacher must be in a valid session **2.**The teacher must be on the dashboard page
Main Flow:	**1.** The teacher is on the exercise setup page. **2.** The teacher indicates that he wants to create an exercise. **3.** The system asks for all the necessary information to create an exercises (language etc), including test cases. **4.** The teacher indicates the test case, if necessary can ask the system for more forms to create additional test cases, "as many as you want". **5.** The teacher indicates that he is finished and records the exercise.
After conditions:	The assignment is created and stored with multiple test cases
Alternative:	**5th.** The teacher decides to discard the exercise. **5a1.** The teacher indicates that he wants to cancel the creation of such an exercise.

Table 3 – Insert assignments with multiple test cases Use Case

RichText Descritpion

Worksheets are fundamental in programming courses, and their assignment hierarchy is vital to students' success. Assignment's descriptions help students understand clearly the problem to be solved, and their clarity is very important. For example, the difference between a well described problem and a poorly described one might improve the students understanding and consequently help them find a solution. Normally, programming worksheets are downloaded in the form of a PDF from an LMS, but CodeInsights offers a mechanism for professors and students to see the assignments from within the system.

Use case Name:	Add image to the assignment description
Primary Actor:	Teacher
Pre-conditions:	**1**. The user must maintain a valid session. **2.** The user must be authenticated with a teacher.
Main Flow:	**1**. The teacher accesses the page of creating an exercise as normal. **2.** The system asks the teacher to provide all the necessary assignment attributes. **3.** When writing the assignment description the teacher indicates he wants to upload the image. **4.** The system asks the teacher which image he would like to add. **5.** The teacher indicates and uploads an image. **6.** The teacher finishes creating the assignment and saves his actions.
After conditions:	The system creates the assignment with an image associated
Alternative:	**6.** The teacher makes a mistake and would like to undo his actions **6.1.** The teacher says to the system he does not want to save the assignment.

Table 4 – Add image to assignment description - Use Case

In past editions of the system, assignment descriptions were simple plain text descriptions, however, teachers remarked that many exercises could not be simply conveyed in this format. A

lot of programming assignments required graphical material, for example, the use of graphics and tables was frequently requested. As a result, the use cases represented in table 11 show all of the stages needed to include such visual elements in the exercise descriptions.

Forgot Password

Teachers and students both have access to the system through a set of credentials, which consist of a username and a standard password. A typical internet user has a large number of accounts on a variety of different websites. The user's credentials may vary significantly depending on the website's password limitations, meaning that they may have a vast number of different passwords across all their accounts at any given time. Auth0 is a company that sells an identity management platform with authentication and authorization services that implements the OAuth2 protocol (among others). Auth0 has a very good reputation in the authentication industry and states that 58% of users admit to forgetting their password frequently, and the average internet user receives roughly 37 "forgot password" emails a year". Because of this, practically every web application must include a forgot password feature.

Despite its importance, the recover password feature was never incorporated in previous versions of the system. Once again, the administrator would directly see or replace the old password with a new password provided by the users. This method became more difficult, if not impossible, once the systems began hashing the user's passwords for obvious security reasons. As a result, in the current version of CodeInsights, the system now includes a password recovery method, whose use case is shown in the table 5.

Use case Name:	Forgot password
Primary Actor:	Teacher
Pre-conditions:	The teacher must be logged of the system
Main Flow:	**1.** On the login page the teacher indicates he has forgotten his/her password **2.** The system asks the teacher for his email. **3.** The teacher provides the system with an email. **4.** The system sends an email with the steps required to reset the password. **5.** The teacher accesses and follows instructions and goes to the reset password page. **6.** The system asks the teacher for the new password. **7.** The teacher supplies the system with his new set of credentials.
After conditions:	The system updates the teacher password
Alternative:	**4th.** Invalid email is entered by the teacher. **4a1.** The system does not notify the teacher that the email is not present in the database.

Table 5 – Forgot password - Use Case

System Integration

The array of tools that instructors use is very vast, nevertheless, an LMS like Moodle is very common among an instructor's toolset. To allow the sharing of information the new CodeInsights version was designed with an Import/Export of entities functionality. Each entity on the CodeInsights such as student's data and their grades will be available via Json text files. The process to download such documents is detailed in table 6

Use case Name:	Export class entity
Primary Actor:	Teacher
Pre-conditions:	**1.**The teacher must be in a valid session **2.**The teacher must be on the dashboard page
Main Flow:	**1.** The teacher is looking at the page that displays all the classes. **2.** The teacher indicates he wants to export all information of a particular class. **3.** The system makes sure he wants to continue with his operation. **4.** The system makes the teacher download a JSON document containing that class's data
After conditions:	A document containing all the classes information (e.g., its students) is downloaded.
Alternative:	

Table 6 – Export class - Use Case

Dashboard Customization

The complexity of the main dashboard appeared to be a problem to some instructors, making them confused and overwhelmed with information. To remove it was idealized a grid layout for this dashboard, where you could move, remove, or add any of the many graphics included in CodeInsights. The table 7 represents the interaction and the sequence of actions to be able to add a graphic to the instructor's main dashboard.

Use case Name:	Add a chart to the dashboard
Primary Actor:	Teacher
Pre-conditions:	**1.**The teacher must be in a valid session **2.**The teacher must be on the dashboard page
Main Flow:	**1.**The teacher selects the option to customize his dashboards. **2.**The system shows you the default template used for the layout of the graphics. **3.**The teacher uses the option to add a graphic to a vacant slot. **4.**The system shows which predefined graphs are available. **5.**The teacher selects the graph he wants to see in the selected slot. **6.**The system adds the graph to the dashboard page. **7.**The system shows the new disposition to the teacher. **8.**The teacher indicates that he wants to keep this new provision. **9.**The system keeps the new layout of the teacher's graphics.
After conditions:	The presentation of the graphics is stored in the system, being associated with the teacher
Alternative:	**7th.** The teacher wants to discard the changes made so far. **7a1.** The system discards the changes and shows the old disposition to the teacher.

Table 7 – Add a chart to the dashboard - Use Case

Processing

CodeInsights supports a wide range of programming languages, from the classic C to the more popular and recent Python. Normally these languages follow an output based evaluation, where they are validated with a series of test cases. However, some languages follow a more visual approach, that can facilitate the student's better understanding of programming.

So, in the lines of the visual aspects of programming, the new version will support the language denominated Processing. Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. Table 8 demonstrates how a teacher can view a processing program that is running on the CodeInsights web application.

Use case Name:	Showing Processing code
Primary Actor:	Teacher
Pre-conditions:	**1.** The intervener must maintain a valid session on the system. **2.** The intervener must be associated with the role of the teacher. **3.** The language of the assignment that the teacher wants to evaluate must be processing.
Main Flow:	**1.** The teacher is on a student snapshot page. **2.** The system, knowing that it is a Processing program, runs it in the teacher browser. **3.** The teacher analyses the code, and manually grades it.
After conditions:	The teacher is able to successfully sees Processing code run on his page
Alternative:	

Table 8 – Showing Processing code - Use Case

Share

Students often make errors in their coding and teachers are responsible for addressing and mending them. When addressing these errors teachers normally make some general suggestions, in order to stimulate their problem-solving capabilities. In some cases, they directly address and propose changes to the code. The system currently allows teachers to edit the student submitted code to fix it. Once the code is fixed they can resubmit the code to re-evaluate it and run it against a set of test cases. This is a process that only involves the teacher and where the student does not participate.

Use case Name:	Start a sharing code room
Primary Actor:	Teacher
Pre-conditions:	**1.** The teacher needs to be logged in to the system. **2.** The initiator must have permission to start a share room.
Main Flow:	**1.** The teacher is on a student's submission page. **2.** The teacher indicates to the system that he wants to create a "share room" with the student from that particular snapshot. **3.** The system creates a link for that share code session. **4.** The system opens a new page on the teacher's browser.
After conditions:	A share room is created with the associated actors
Alternative:	

Table 9 – Start sharing code room - Use Case

However, several teachers noted that they felt a need to show such code changes to students in real-time in order to explain them better. Thus, a code sharing system was one of the new functionalities conceived for CodeInsights. With a code sharing system, it would be possible for teachers to share the code they are editing with either a class or a single student. For example, if a teacher identified a class common error he/she could address it directly in class by asking all the students to access the share code mechanism. Table 9 details how a teacher can create a share code room.

BlackList Keywords

In programming it is quite common to have an external library that implements a series of methods and features that make life easier for programmers. This is very beneficial for experienced programmers who don't need to spend time implementing basic and simple functions. Using the same example of finding the maximum value in a list, there are several libraries that have already implemented a method that achieves this job. For example, in Java and using the java.util.Collection library, programmers can perform this search through the max function. This applies to every language whether that is C or Java or any other.

External libraries are quite good for saving time, but for students whose main goal is to learn, it can be quite detrimental. Students can use these pre-implemented functions and solve exercises very quickly without stimulating their problem-solving capabilities. However, they are just making their problems bigger by not addressing them and ignoring them. For this reason, it was necessary to put in place a system that would allow teachers to block certain keywords and to mark submissions in which they would be used, for future analysis by the teacher.

The table 10 shows the use case utilized to describe the operation of adding black-listed words.

Use case Name:	Add keywords to black-list
Primary Actor:	Teacher
Pre-conditions:	**1.** The user must maintain a valid session. **2.** The user must be authenticated with a teacher. **3.** The teacher needs to have already created an exercise.
Main Flow:	**1.** The teacher accesses the page of the exercise that he wants to use a black-list. **2.** The teacher tells the system that he wants to add a word to the blacklist. **3.** The teacher writes the word. **4.** The teacher indicates that he intends to complete this operation.
After conditions:	The system add the correct keyword-assignment relation to the database
Alternative:	

Table 10 – Add keyword to black-list - Use Case

RichText Descritpion

Worksheets are fundamental in programming courses, and their assignment hierarchy is vital to students' success. Assignment's descriptions help students understand clearly the problem to be solved, and their clarity is very important. For example, the difference between a well described problem and a poorly described one might improve the students understanding and consequently helping them find a solution. Normally, programming worksheets are downloaded

in the form of a PDF from an LMS, but CodeInsights offers a mechanism for professors and students to see the assignments from within the system.

In past editions of the system, assignment descriptions were simple plain text descriptions, however, teachers remarked that many exercises could not be simply conveyed in this format. A lot of programming assignments required graphical material, for example, the use of graphics and tables was frequently requested. As a result, the use cases represented in table 11 show all of the stages needed to include such visual elements in the exercise descriptions.

Use case Name:	Add image to the assignment description
Primary Actor:	Teacher
Pre-conditions:	**1**. The user must maintain a valid session. **2.** The user must be authenticated with a teacher.
Main Flow:	**1**. The teacher accesses the page of creating an exercise as normal. **2.** The system asks the teacher to provide all the necessary assignment attributes. **3.** When writing the assignment description the teacher indicates he wants to upload the image. **4.** The system asks the teacher which image he would like to add. **5.** The teacher indicates and uploads an image. **6.** The teacher finishes creating the assignment and saves his actions.
After conditions:	The system creates the assignment with an image associated
Alternative:	**6.** The teacher makes a mistake and would like to undo his actions **6.1.** The teacher says to the system he does not want to save the assignment.

Table 11 – Add image to assignment description - Use Case

CodeInsights provides a feedback mechanism to the student, yet it is based on email messages making the communication an outside element of this system. Another missing functionality of the system is the ability to request help from the instructor. So, combining these two requirements we decided that the best approach was to create real-time chats, enabling better communication and understanding between student and instructor. The table 12 describes the task of creating chats.

Use case Name:	Start a chat discussion
Primary Actor:	Teacher
Pre-conditions:	**1.** The teacher needs to be logged in to the system. **2.** The student also needs to be registered in the system. **3.** The initiator must have permissions to start a chat.
Main Flow:	1. The teacher is on a student's submission page. **2**. The teacher indicates to the system that he wants to create a "chat" with the student from that particular snapshot. **3.** The teacher sends a message. **4.** The system creates the necessary association. **5.** The teacher sees his message being sent where the chat is shown.
After conditions:	A chat is created with the associated actors
Alternative:	

Table 12 – Start chat - Use Case