```
//...

@Entity
@Table(name = "images")
public class Image implements DomainEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private -- select an option -- id;
            Integer
    @Column(String
    private int

    private Integer width;
```

Tutor for learning based on multiple choice questions

Pedro Miguel Comparada Reganha

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. António Manuel Ferreira Rito da Silva

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. António Manuel Ferreira Rito da Silva
Member of the Committee: Prof. João Carlos Serrenho Dias Pereira

January 2021

Acknowledgments

I did not walk the straightest path to reach this finish line. After being captivated with the concept of working, start earning money and learning new subjects every day on the job, I became less motivated to conclude my studies. In addition to that, life as a way of throwing you curveballs when you least expect that make you shift gears without much control. Until reaching this conclusion, I had one failed thesis topic, which I barely worked on, and another which I concluded the first part, but did not drive it through. Both of those failures are mine and mine alone, and with a pondered decision, I moved on. I want to begin my thanks to both of my original supervisors for the time spent working with me.

I would like to thank my parents and my brothers for never stop believing in me. They always pushed me to thrive for excellence and work hard to achieve my goals. It is crucial to me as well to thank my girlfriend Mariana, who was always there for me and impelled me to push onward this goal that I had and left behind along the way. Without them, I would probably not be writing this Thesis right now. Thank you.

After setting for myself the goal of finishing my Masters finally in 2020, I had no idea what this year had in store for us. I can not thank enough for the guidance and patience demonstrated by my supervisor António Rito Silva. Working and wrapping the Masters was not an easy task, and this support was vital. Also, this would not have been possible without the support of my company, Syone, and my client, Godtlevert. In combination with my fantastic team members, they gave me space and assistance to complete this education.

Also, I would like to thank my friends without naming anyone in particular; we have been through thick and thin since the start of this journey. It is incredible to see that we can still rely on one another.

Finally, I would like to thank my grandmother, one of the persons in my life that always painted my road to greatness. She never stopped believing in her ”engineer” grandson. As I am writing the final details of this Thesis, I learned of you passing away. Let this work also serve as a thank you for all the years you encouraged me to be better and aim higher.

To each and every one of you – Thank you for everything.

Abstract

Education is in constant evolution. Lectured content has been changing throughout the years as well as the techniques and tools to deliver them. Quizzes Tutor is a solution to serve both the teachers and the students using multiple-choice questions.

To support teachers with the challenge that is teaching programming Quizzes Tutor platform was extended to encompass code type questions, adding to the existing multiple-choice ones.

This thesis starts by analysing and understanding other solutions that perform programming code question and answer analysis, especially those that do it in a quiz-like scenario. With that in mind, looks into the existing platform to solve the two major problems: code restructures to allow multiple question types and adding the new code question. This paper describes the steps taken, and the thought process involved in the construction of this extension. It also presents a performance analysis of the new solution, comparing it with the previous one, finalising with an evaluation of the effort of adding a novel question type to the new solution.

The thesis developed allowed for a better code structure with elasticity in mind making it easier for future developers to improve upon this solution or add new code questions for new use cases.

Keywords

Programming; Learning; Teaching; Quizzes

Resumo

A educação está em constante evolução. O conteúdo lecionado tem mudado ao longo dos anos, assim como as técnicas e ferramentas para disponibilizá-lo. O Quizzes Tutor é uma solução web para ajudar os professores e os alunos por meio de questões de múltipla escolha.

Para apoiar os professores no desafio de ensinar programação, a plataforma Quizzes Tutor foi estendida para abranger questões de programação, adicionando-se às existentes de múltipla escolha.

Esta tese começa por analisar e perceber soluções existentes que utilizam perguntas de código bem como a sua avaliação automática, tendo uma especial atenção naquelas que o fazem num cenário semelhante a quiz. Com isto em mente, analisa a plataforma existente para resolver dois problemas principais: reestruturações de código para permitir vários tipos de perguntas e adição de novas perguntas de código. Este artigo descreve os passos dados e o processo de pensamento envolvido na construção desta extensão. Também apresenta uma análise de desempenho da nova solução, comparando-a com a anterior, finalizando com uma avaliação do esforço de adicionar um novo tipo de questão à nova solução.

A tese desenvolvida permitiu uma melhor estrutura de código com elasticidade em mente, tornando mais fácil para os futuros profissionais que trabalharem em melhorar esta solução ou adicionar novas questões para cobrir novos casos de uso.

Palavras Chave

Programação; Aprendizagem; Ensino; Quizzes

Contents

List of Figures

List of Tables

Listings

Acronyms

UI	User Interface
UX	User Experience
DTO	Data Transfer Object
DDD	Domain Driven Design
IDE	Integrated Development Environment
SDK	Software Development Kit
MOOC	Massive Open Online Course
FE	Frontend
BE	Backend
LOC	Lines Of Code

1 Introduction

Contents

Every student has a different rhythm, difficulties and goals. Education needs to grow to be tailor-made for each one of the students. These do not apply only to the university level but focusing on it we see that many students are not ready for the hardship that is to come on what will most likely be of the last couple of years studying formally. Here the role of the teacher is, of course, to expose a given subject but also guide the next minds of our generation. A problem we see is like stated before different students exhibit different learning velocities.

E-Learning came to transform how content can be delivered, using the evolving network technology to deliver such contents. By definition, E-Learning is the usage of electronic technologies to create learning experiences [1]. E-Learning can take many forms, depending on the goal or subject, we can have multiple tools working together. Standalone courses are one variation on e-learning, commonly created to be taken on by a solo learner — self-paced and not requiring interactions from external elements, either teachers or classmates. Many of these courses exist on mobile devices. Mobile learning as been growing with the increasing adoption of mobile phones. It is normal to walk on the street and see people trying to improve their English using, for instance, mobile applications built for that. Furthermore, learning games and simulations have also gained considerable space in e-learning. They are ranging from simple quizzes to advanced simulations that can help us understand complex problems. Another variation of e-learning is Social Learning, where learning is encouraged through interaction with a community of specialists and other students. Wikis, blogs and podcasts are a way to disseminate the community's knowledge to create discussion and learnings [2]. Lastly, virtual-classrooms are the equivalent of a typical classroom but where everything happens in the ”cloud”.

Both Bodzin and Cates [3] and Santally and Raverdy [4] noted that in comparison to the ”traditional” approach, e-learning could take contents to the next level. It provides their learners with more materials and different ways to look at the same topics, promoting learners with increased learning effectiveness. Now they can get custom-tailored education. The best to take from it is that it shortens time and space constraint, that would happen in a regular class, and in turn, provides learners with more opportunities to learn with more prominent spontaneity [4]. E-Learning does appear to present a strong case with these advantages, but there is still no actual data to prove that it can always lead to a better learning outcome.

However, even when not encouraged students already rely on multiple online sources, depending on the topics, they rely on YouTube or more specialised channels like Udemy, Coursera or even learning sites, some even from other universities lectures. However, there is a problem with this. The materials accessed by the students are not always correct and might be insufficient. Furthermore, for an already confused student, these insufficiency's and errors might pass by unnoticed. In recent years has also arisen projects from certain universities that record videos of the relevant subjects and then shared them to make the knowledge share more focused and repeatable. These videos also help counteract the effect of ad-hoc training.

Another problem that we can see with this is that the students can use these as a learning mechanism, yet they only expose a given topic. In order to actually learn and understand is essential to practice, make errors, and learn with them. Most subjects have an exercise compilation that students can use, but not always provide a solution making the students study less effective. Another possible difficulty is that the given problems are relatively limited, with the use of the web, the distribution of said materials has increased to support the students' needs.

Having an online tool to train, learn and put themselves to the test is essential nowadays. This tool can have different difficulty levels and needs to be easily extended, and even one simple problem could be parametrised so that it is actually many more exercises than just one.

These online tools became even more relevant during 2020 when COVID-19 became a world-wide pandemic forcing regular teaching and learning methods to adapt—having to go from a fully presential system to fully remote or partially remote demands a better online presence and mechanisms for spreading knowledge and practising it.

Nonetheless, even without the pandemic, tools like these are essential as people in our busy world also start to resort to lots of learning materials, even universities are going for the fully online courses, these come with challenges but have a significant advantage of reaching more people. E-learning is a big bet nowadays and might even replace universities entirely if they do not evolve.

Many of the current solutions exist for a specific thesis project or are custom-made to a given classroom need. Introducing Quizzes Tutor [5–7] a quizzing and assessment tool that allows teachers to create a multiple-choice question bank and then use it in multiple assessment and quizzes. Its development started as an effort of project IMPRESS[1] [8–10], being like a custom-made tool that would fill the necessities of a project but always ready to reach other teachers and classroom needs. However, it did not die here and continues to be improved.

Quizzes Tutor exists to ultimately achieve a flipped classroom, using e-learning tools to accomplish it. When interacting in classes with Quizzes Tutor, the learners' process would be to prepare the next class and even challenge themselves using the platform. Attend the class to explore concepts learned and put them into practice. Finally, after class, students should extend their knowledge and recur to Quizzes Tutor as a knowledge validation tool. Quizzes tutor is a multiple-choice only platform. The acceptance of the tool started growing amongst the students. This lead to a demand for incorporating code questions so that the students could use to challenge themselves. Creating there a place for them to practice and learn and eventually even be graded, removing, for instance, the need for the classic exams where the student would write code on paper. Multiple solutions were already posed to this problem of having an in-class tool for automatic assessment of programming code. However, as stated in a review of these

[1] https://impress-project.eu/

systems [11], they are created with a limited scope or lifespan, as for each new thesis or course. Also, new tools tend to be created to fill the needs or prove a point, which ends up being discontinued.

The challenge with adding programming code questions lies with the analysis and evaluation part. On the most involved end of this spectrum, we have the development of a full piece of software. The full development consists of a simple interface that would be a simple code editor. However, we would have to either run the code or run a static analysis of the code to understand its correctness for evaluation. The first one would be a simple solution, yet for a scalable software solution, where we have multiple students answering, can become intricately complicated. On the other side of the scale, we could have code completion based on a list of options, the evaluation here would be: is the correct option selected?

Quizzes Tutor aims to be a tool that can provide an enormous scope, with the capability of encompassing multiple use cases. This work will be adding the possibility of adding new types of questions, particularly some programming code question. Quizzes Tutor is built considering best practices and using development methodologies like Domain Driven Design (DDD) [12]. Adding to this is also focusing its eyes on modern technologies: Docker, Git, VUE.js with Typescript, among others. That being said, this makes it a tool for learning even more impressive than the quiz platform that it is. Students will also be challenged to contribute and develop on top of this software, having the source code of the tool serving as a base for teaching and knowledge sharing in specific courses. This challenge is advantageous on two different notes. Firstly, it makes them work on a full-fledged application to better exercise their ability to work in larger applications. Secondly, it will keep the project alive and counteract the projects that exist only to fill one specific use case or a specific course that eventually dies.

1.1 Objectives

This thesis's main objective is to extend the existing Quizzes Tutor solution to increase its potential for more use cases than the multiple-choices questions, which were its first use case.

This new extension will add a new quiz question type, more specifically a code question based on fill-in answer. An extension to the existent domain model will be required to allow multiple question types for the quizzes to accomplish this primary goal. Finally, as part of these thesis objectives, the new solution should provide a flexible structure that allows for a reduced effort to add new question types. This new solution should not have significant performance impacts when comparing with the previous one.

Performance comparison test will evaluate these goals to ensure that performance is not meaningfully affected. Apart from that, a different programming code question will be measured to ensure that effort is indeed low when adding new question types.

1.2 Contributions

The major contributions to the Quizzes Tutor solution are the following:

- Restructure of code in order to allow multiple types of questions.
- Introduction of code questions, fill-in based and order problem (introduced during the evaluation phase).
- Ease the future introduction of new types of questions.

1.3 Organization

This document is organised in the following way:

- Chapter 2 is the related work, which gives, initially, an overview of the advantages of using quizzes for learning. Then dives into the possibilities that can arise by the introduction of multiple question types. Concluding by exploring the introduction of programmatic code questions exposing both existing solutions, challenges of introducing them, and some solutions.
- Chapter 3 denotes the solution overview. It starts by going over a background of the existing Quizzes Tutor solution to know it better. It goes over by the requirements of the solution, to have an understanding of the expectations and restrictions. Finally, it goes into the de facto solution in the closing two sub-sections, the multiple question types and the new code question. The multiple question types evaluate the DDD approach done to understand the transformations required to add new question types and exhibits the actual changes. Then, having multiple types of questions, the new code question chapter concludes with the process that culminated in the addition of this new question type, whilst considering the requirements.
- Chapter 4 corresponds to the evaluation of the solution. This chapter begins with the performance evaluation, comparing the initial solution against the new solution. It performs different tests to cover the principal cases (e.g. question creation and quiz answering). Finally, it portraits an appraisal of the facility of adding a new question type, in this case, a code question.
- Chapter 5 is the conclusion, which contains a summary of the work performed from a critical judging standpoint. It wraps-up with the future work to be accomplished.

2 Related Work

Contents

This chapter performs an analysis and exhibition of the state of the art of quizzes as tools for learning exploring multiple relevant aspects.

Firstly presents both benefits and reasons to use quizzes as a learning tool, exposing multiple experiences and analysis in the benefits of using these kinds of tools to help students retain knowledge and learn.

After understanding the reason to include this kind of methodologies into learning processes, it explores how to make learning/teaching programming a more prolific effort. It starts by analysing some methodologies and tools used. Then, proceeds into focusing on the tools more specifically, the automated assessment of programming assignments. Although these tools are not always correlated with a quizzing functionality, they show-case interesting scenarios that could easily be ported into the quizzing world as some previously performed. After that analysis, it focuses on some concrete use cases of programming code questions in quizzes or similar.

Finally, it presents an overview of possible quiz question types, how they differ, and some of the challenges that might occur when implementing them. It uses these question types to explore even more relevant use cases for programming questions. This comparison is made by clearly exposing some of the advantages of using such question type as a programming code question. This analysis originates inspiration both for the work developed as well as for possible future work.

2.1 Quizzes for learning

As stated in 2006 by Roediger et al. [13], tests have a positive effect on memory and knowledge retention. When tested on a given material, the testee can retain that knowledge for a more extended period. They even go further, by illuminating that having that evaluation is always more beneficial, even if without feedback, then extra learning materials. That being said, Butler and Roediger [14] note that educators should always provide feedback when using multiple-choice quizzes.

It is common for university courses, and not only, to overlook evaluations as a learning tool. In many situations, evaluations occur infrequently and are typically treated as bothersome by both the faculty and students alike [15]. This negligence is rather gloomy and misguided. Having a systematic way of testing the students more regularly will pressure students to study more regularly rather than just on the days before exams. Roediger et al., have noted that whenever students are questioned on a given subject and successfully recall or recognise it, they will recollect it better in the future. This phenomenon is known as the testing effect. Changes of remembrance would be lower when learners do not evaluate their knowledge on a given subject.

Tests are, of course, a great way of gauging the students' knowledge, both from the student's standpoint as well as the teacher. Nevertheless, they are excellent learning tools should not be neglected, where even exams without feedback can improve students knowledge as seen before. Many of these tests described before can have repercussions on the final grade, even to incentivise the students to take them. Nevertheless, even in low-stakes testing, benefits can be gained. In 2011 McDaniel et al. [16], introduced quizzing as a technique to improve learning and retention of a given subject, in this specific case, in a middle school classroom. This evaluation with low- or no-stakes proved considerably efficient in keeping the students aware of the course curriculum, allowing them to achieve even more outstanding grades.

One relevant aspect to notice is that sometimes students choose to ignore evaluations with low stakes, especially in the higher education sector. It is then imperative that we motivate our students to learn. As seen before quizzes can provide a great way to maintain knowledge, so we need to empower them to captivate the students. One study showed some efficacy in providing a gamification factor in the quizzes [17]. Students were happier and motivated to finish and participate in the quizzes, increasing this way their engagement levels.

Correlated with students' motivation, an inverted, or flipped classroom can be a great way to capitalise on that motivation. In an inverted classroom, the learnings start at home, before the actual class. Students learn by studying videos, quizzes and other learning relevant materials. This individual and initial study allow the liberation of the classroom to perform learner-centred activities and problem-based learning. In 2013 Mason et al. [18] made a comparison between traditional classrooms and inverted classroom in an engineering course, with excellent results. They evaluated three main areas: content coverage, student performance and student evaluation of the inverted classroom. The inverted classroom covered more class content; they had better results in quizzes and exams, and finally, besides the initial shock of this new methodology students adapted quickly and found the format useful. Using these new methods combined with quizzes are fabulous, students can explore even more material and achieve tremendous success. Students that are becoming independent self-learners that can access their knowledge independently and adjust study as needed.

2.2 Programming coding questions

It is customary for students pursuing an engineering course, but not only, to have at least one introductory programming course. They are deemed both a relevant tool nowadays and help develop logical and reasoning skills and improve problem-solving abilities.

Although many papers tend to use high failure rates as a reason to improve on programming education, these are relatively low. They are globally close to 30% in introductory classed [19, 20].

Despite that, it is still noted by Lahtinen et al. [21] in a survey conducted in 2005 that students feel unprepared. Programming for them is not only challenging because of the abstract concepts, but also because of problems related to the actual code construction. Students state that it is more relevant to explore and do the programming by themselves to foster these learnings. Teachers can use the right tools, materials, and approaches to guide them in gaining knowledge and skill constructions.

The teachers can use multiple approaches to achieve the ultimate goal of teaching programming. Across the years, there has been a development of multiple techniques and tools to help students succeed in this field. Multiple surveys were done in the area [22, 23], below are listed some techniques [23]:

- Collaboration: the usage of activities that encourage student collaboration is great for knowledge transfer between peers. For example, collaboration in class (problems and labs) or collaboration outside of class like coding projects.
- Content change: keeping the material up-to-date is an excellent way to keep the students engaged and motivated.
- Bootstrapping: consists of easing into the programming world, instead of starting a complex first programming language. A good example would be starting with visual programming tools like Scratch or having introductory classes.
- Grading schema: inverting a typical pattern that gives a high value to the final exam and instead students obtain the final grade for the year from programming assessments or quizzes.
- Relating to reality: either by using games or more media like materials, that usually are more visual and relatable, can increase student engagement.
- Support: increased teacher hours, forums or other support channels, among others.

More relevant for our approach then the techniques that focus mostly on human-to-human interaction are the tools that can be used to increase the students' learnings, more specifically [22]: visualisation tools, automated assessment tools, and programming environments. Complex problems can immediately become simpler through the usage of a simulation or a scheme. For example, to understand how algorithms work is useful to have a tool that simulates its execution. A visualisation tool does just that. We can have visualisation tools that execute the algorithm ”step-by-step” to explain the execution flow. Another excellent example of a visualisation tool is an actual programming tool like Scratch, for instance, which is an excellent tool to start understanding programming concepts and can be very beneficial even from a young age [24]. Teachers can use visualisation tools as a great way to keep the students engaged and achieve better results when it comes to explaining a given issue.

Nonetheless, students might still feel lost or might not be sure that what they are building is correct; presenting automated assessment tools. This tool can be used both for the benefit of the student and the teacher. When thinking about the student, they can be used to ensure that what they are doing is correct.

The correctness should never be a simple evaluation of whether the code runs, and students tend to be unable to think on every scenario. Having a tool that tells them possible failures that they might have missed is essential for the learning process, and having this feedback soon is fantastic. On a similar note, for teachers having these kinds of tools make their work possible. If they had to evaluate all the code of all the students individually would be catastrophic also, it might create inconsistencies in some correction, mostly if multiple teachers divide the work amongst them. With the automated assessment tools, all code analysis is using the same criteria.

Finally, Pears et al. [22], note the relevance of programming environments. Regardless of the seniority level and know-how, all programmers work within a programming environment composed of the tools they require to achieve their goals. For example, if one develops with Java, they must have the Java Software Development Kit (SDK), they can have a full-fledged Integrated Development Environment (IDE), like Eclipse[1] or IntelliJ[2], or a simple code editor and a terminal. Having an IDE is an excellent addition to your programming environment as it will: organise your project, have advanced language-specific editing features, and support tools (like visualisation tools for debugging or code analysers to prevent errors). However, the complexity of an IDE can be a disadvantage in introductory courses, as they can lead to a lack of understanding of how everything is working.

The automated assessment tools are the most interesting to analyse even further, as we will be adding code questions, and there is a need, of course, to evaluate them and expose their correctness. During the remainder of this section, we will analyse such tools' requirements and how they can be used in Quizzes Tutor.

2.2.1 Automated Assessment of Programming Assignments

As stated previously and noted since 1994 [25], it is not possible to evaluate programming assessments in an impartially and systematic way without automation support. Automatic tools can perform both static and dynamic analysis, with semi-automatic or fully-automatic approaches. In 2005, Ala-Mutka [26] did a survey of automated assessment approaches for programming assignments, resulting in a division and exposure of each procedure. This section will go through them and explore the existing tools that fit these profiles to understand their approaches.

1. Dynamic Analysis

   A dynamic analysis consists of testing and evaluating a program by executing it in real-time. The concept on-itself is a simple one, execute the code and check it. However, the execution of the code can bring problems. As noted by Ala-Mutka [26], either by a bug in code or a student with malicious intent, can cause the evaluation system to malfunction. Because of this, the automated

[1] https://www.eclipse.org/
[2] https://www.jetbrains.com/idea/

dynamic analysis must happen in a secure environment, a sandbox. When thinking of dynamic analysis, Ala-Mutka [26] notes four main approaches: functionality, efficiency, testing skill, and special features.

2. Static Analysis

   The static analysis consists of the evaluation of code without executing it and just ”reading it”. This process is excellent to understand the code structure and whether the code adheres to best practices. When thinking of static analysis, Ala-Mutka [26] notes four main approaches: coding style, programming errors, software metrics, design and special features.

3. Semi-Automatic vs Fully-Automatic

   Automatic assessment and evaluation can sometimes be considered incomplete as the machine's feedback be insufficient pedagogically speaking when compared to an instructor. Even in the same course, it is common to use both methods, for instance, fully-automatic for small programming tasks, that usually focus on programming basics. On the other end, for larger assignments, the semi-automatic approach usage allows for the combination of both worlds.

4. Formative vs Summative

   Another perspective on the automatic assessment is the feedback given. It can be summative, where the students only receive a grade, not being clear where to improve. Alternatively, on a different perspective is the formative one, where students can see the results of their submission and in some tools even re-submit the answer until they are pleased. This last scenario provides for an exciting learning opportunity.

2.2.2 Programming Code Question in Quizzes

The previous section focused mostly on the automatic evaluation part, and while that is hugely relevant, it is crucial to focus on how other investigators have been using programming code questions in a quiz-like scenario. There is a considerable amount of solutions that incorporate code questions in a web/e-learning scenario.

Starting with the most programming similar situation is the quizzes or online exams which essentially require a learner to develop a piece of software inside the browser. CodeRunner [27] and and CS Circles [28] are examples of these tools. CodeRunner started from the challenge that is evaluating hand-written code. On 2016, Lobb and Harlow [27], developed an extension to Moodle, to evaluate students programming skills. It can perform style checking and dynamic analysis. For security, the student code's execution takes place on a separate machine, called the sandbox server. The tests' output is then displayed to the student, which can use it to improve upon the existing code. To decrease server calls some validations are done directly on the client: style checking, and enforcing or restricting programming constructs. Regarding the second validation, if for example, a question requires a while clause, and that is none present, there will be no submission to the server. CS Circles [28], built in 2013, is more of a learning platform, which uses questions embedded in their learning materials. Is constructed on top of WordPress[3]; for that, it uses CodeMirror [29] to mimic the code editor and adapted from the Mooshak [30] it uses ”safeexec” which is used to run the code under evaluation.

Another common scenario when considering programming questions in quizzes is evaluating the output given a particular input. These types of questions when parameterized are fascinating since it forces the student to think about the problem, trial-and-error makes no sense when each incorrect input would require a restart from scratch to submit again. QuizPack [31] and QuizJet [32] use this approach (section 2.3.6 explores them in more detail).

Both of the previous problems require a more dynamic approach, especially on evaluating the answer part. Regardless of that, there are more straightforward approaches taken by multiple programming quizzing systems, that whilst being more limited are considerable easier to setup and usually require fewer resources. The Parsons Problem [33] is an example of a said question where students must order blocks of code to form the correct answer. The next section takes a look into multiple quizzes question types and their integration with programming code questions.

2.3 Types Of Questions

As seen before, quizzes are a practical way for a student to learn, retain knowledge and gauge the current level on which he currently is. However, most quizzes are based on multiple-choice, which makes them limited to what they can do. Quizzes Tutor suffered from this, making it hard to incorporate other question types and more precisely, fill new use cases. With the platform’s extension, we will be able to add new question types easily, like the ones displayed below or incorporate other ways to gauge students’ knowledge that might even replace written evaluations.

Currently, Quizzes Tutor already supports quizzes for multiple different subjects, supported by a question bank, filled by the teachers and the students. Quizzes Tutor allows for students to create auto-generated quizzes from specific topics or answer quizzes created by the teachers. It has been built to be easily used in classes, with QR Code support, or exams with time constraints and question rules (e.g. after answering a question, a student cannot go back). Recently added tournaments can be managed by the students encouraging to have more learning interaction between each other. Adding to that, students can access statistics related to their performance on a given course. All these capabilities are exceptional and can be powered even further with different question types that allow the students to be

[3]https://wordpress.com/

put to the test with several scenarios.

In 2017, Chauhan and Goel [34], performed an analysis of quizzes in Massive Open Online Course (MOOC) platforms. MOOC have quizzing capabilities either associated with their video materials and independent quizzes. Chauhan and Goel [34] listed both question types and relevant features. This list and analysis is an essential starting point to understand the possibilities of question types. It is also possible to compare it to other analysis and understand the application for programming code questions. The current section will mostly focus on the question types presented in this study and others alike.

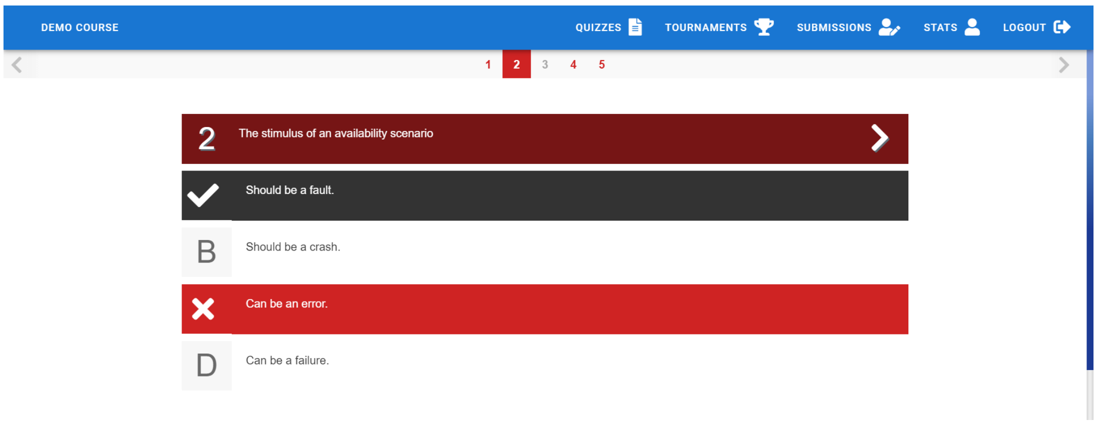
This section will present the possibilities of extending the tool by allowing multiple question types, listing some question types, and their applicabilities with code questions.

2.3.1 Single Selection

Chauhan and Goel [34] identified that all primary MOOC have the single selection question type amongst their question types. These single selection questions divide themselves into TrueFalse question and Multiple Choice Question.

2.3.1.A Multiple Choice

The multiple-choice is the most classical question type, one has multiple options to choose from (usually four), and only one is the correct option. This question type is the most common question type easily found throughout the literature [34–37]. One common way to use this question type with an increased difficulty setup is with the Single Best Answer where all answers might be correct, but there is one that is even more correct. This question adaptation can be great to differentiate high and low performers in quiz format [36]. On fig. 2.1 we can see an example of a multiple-choice question answered incorrectly in the Quizzes Tutor platform.



2 The stimulus of an availability scenario

Should be a fault.

B Should be a crash.

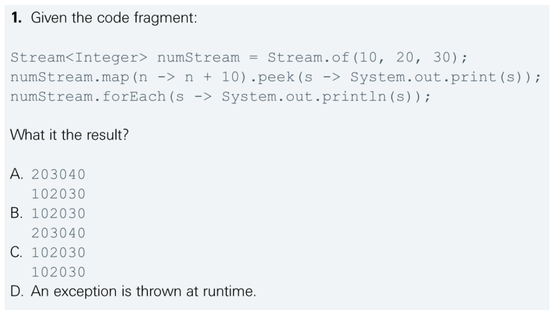
Can be an error.

D Can be a failure.

Figure 2.1: Multiple Choice Question Answered Example (Quizzes Tutor Platform)

When it comes to evaluating a multiple choice question answer correctness, the approach usually is straightforward. As used in Quizzes Tutor [5, 6] and other platforms, the strategy is to compare the answered option against the correct option; if the options match then the answer is correct, and the examinee gets full marks. Another possible approach for the evaluation is to give partial marks to some options. EdX [38] allows for this approach, where one can give a partial grade to the examinee if answers an option that while not being entirely correct is not altogether incorrect. The partial credit idea is to motivate learners who have comprehended some of the course materials and provide a score that demonstrates their progress and effort.

From a programming code question perspective, the multiple choices questions are not our first thoughts. However, given their simplicity of both evaluation and setup, are questions used by multiple certification entities like Oracle[4] and Microsoft[5]. On the study guide for the Oracle Java 11 exam [39], we can see examples of multiple-choice questions being used for programming code questions. Usually, when this scenario is used, it focuses more on evaluating the program rather than completing it, but as seen on that material, it can be used as both. Figure 2.2 gives an example from that document, demonstrating the usage of programming code questions as multiple-choice questions.



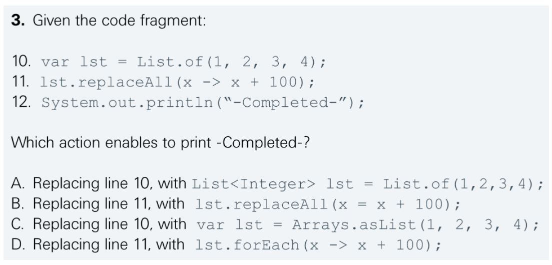
1. Given the code fragment:

```
Stream<Integer> numStream = Stream.of(10, 20, 30);
numStream.map(n -> n + 10).peek(s -> System.out.print(s));
numStream.forEach(s -> System.out.println(s));
```

What it the result?

A. `203040`
   `102030`
B. `102030`
   `203040`
C. `102030`
   `102030`
D. An exception is thrown at runtime.

(a) Question that requires code examination and understanding its execution



3. Given the code fragment:

```
10. var lst = List.of(1, 2, 3, 4);
11. lst.replaceAll(x -> x + 100);
12. System.out.println("-Completed-");
```

Which action enables to print -Completed-?

A. Replacing line 10, with `List<Integer> lst = List.of(1,2,3,4);`
B. Replacing line 11, with `lst.replaceAll(x = x + 100);`
C. Replacing line 10, with `var lst = Arrays.asList(1, 2, 3, 4);`
D. Replacing line 11, with `lst.forEach(x -> x + 100);`

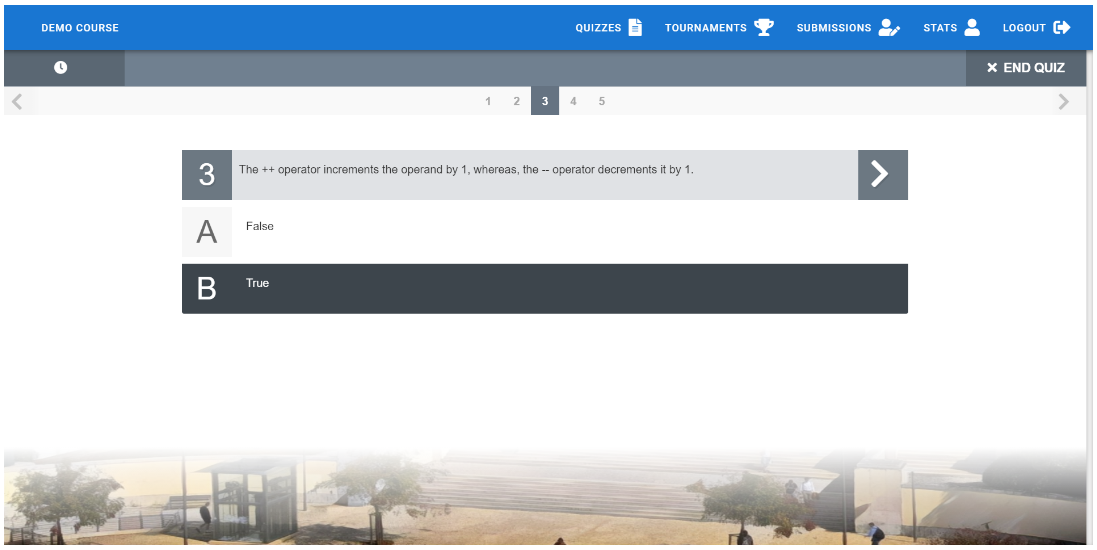
(b) Question that focus on fixing the code to achieve the correct result

Figure 2.2: Oracle Java 11 Certification questions that use multiple-choice questions

2.3.1.B True or False

True or false questions are another type of single selection question, where there is usually a statement, and the answer is either true or false, being that only one is correct. True/False questions are one of the most manageable formats to write and can encompass a significant number of contents with few questions. However, it is sometimes found a question type with low fidelity as there is a high chance of guessing associated with it [36]. Figure 2.3 depicts an example of a true or false question. It uses the same interface as the multiple-choice (section 2.3.1.A) question in the Quizzes Tutor platform.

[4]https://education.oracle.com/certification
[5]https://docs.microsoft.com/en-us/learn/certifications/



3 The ++ operator increments the operand by 1, whereas, the -- operator decrements it by 1.

A False

B True

Figure 2.3: True or false question example (Quizzes Tutor Platform)

Regarding evaluation the situation is similar to Multiple Choice questions. The answer given is compared to the correct one and if they match then the examinee receives the full marks. Also, using this type of questions for programming code questions might be restrained, but as the Multiple Choice questions, they can be used. The scenario here would usually be a piece of code followed by a statement, that would either be true or false.

2.3.2 Multiple Selection

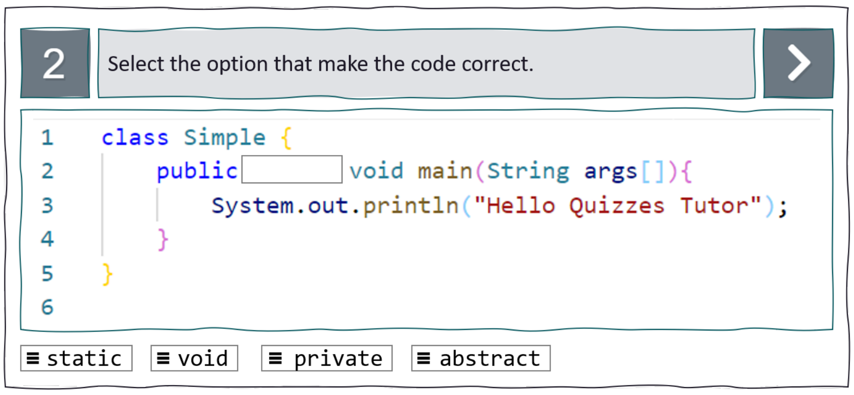
Multiple selection questions, also known as Checkbox questions, require multiple answer selection to form an entirely correct answer. This question is common in MOOC [34, 40]. Johnson [40] used them as a way to challenge the participants to think about all the answers instead of having the best answer given in a plate. In terms of the Quizzes Tutor's implementation, this can be precisely the same as a multiple-choice question, with the difference that would allow multiple selections.

In multiple selection questions, students select one or more options from a list of possible answers. To answer the problem correctly, a student must select all of the options that are correct, and none of the options that are incorrect. The grade attributed to each responder can vary. EdX [38] can support a simple system like if all correct are selected and no incorrect are correct then you receive the full marks, other than that you gain 0. Other than that they also support a more pondered approach, with partial points. The first method, denoted as ”every decision counts”, takes in the number of options (n) and each correct choice they make earns them $1/n$ points. A correct choice is considered either not selecting an incorrect option or selecting the correct one. The second method, named ”by halves”, basically each error made by the learner will half is final score in half, if more three errors are made then the learner receives no credit for the problem.

This question type usefulness in terms programming code question is identical to the Multiple Choice question. The difference here would be that multiple responses would fulfill a given answer. This normally is used in a programming scope context but goes more into the theoretical side of programming, which is already easily supported in Quizzes Tutor [5, 6].

2.3.3 Fill in the blank

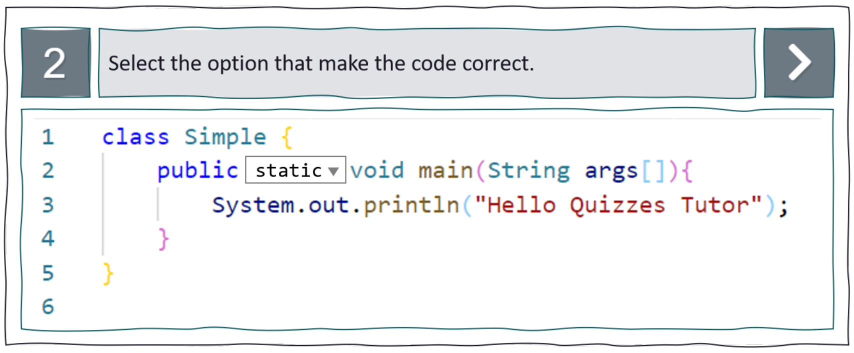
Fill in the blank is one of the most popular types of questions in testing. This question type is widespread when teaching new languages, but can have an exciting application in programming teaching as well. The question itself consists, typically, of a piece of text with some words or sentences removed. Students must then complete with the correct removed words or sentences. It is interesting how this specific question type is very analysed and used when considering automated question generation, for example, from textbooks [41–43]. There are multiples possibilities when thinking about this question type. We can drag and drop, open text to write, or select from a drop-down.



Select the option that make the code correct.

```
class Simple {
    public          void main(String args[]){
        System.out.println("Hello Quizzes Tutor");
    }
}
```

(a) Fill in the blank - Matching Items mockup



Select the option that make the code correct.

```
class Simple {
    public static void main(String args[]){
        System.out.println("Hello Quizzes Tutor");
    }
}
```

(b) Fill in the blank - Multiple Options Answer mockup

Figure 2.4: Fill in the blank question mockups

This type of question is a great candidate for using with code questions, where the student can see the code and act on it right where the code is. When using this question type in a programming code scenario, the normal text would be replaced by the code in any given language, and if possible that code would be with the correct highlighting for easier recognition. As an example of this implementation, we have the work of Funabiki et al. [44, 45] which approached this problem with an open answer box. Their solutions were created for Java teaching; the solution was fascinating in the blank generation. They were able to generate the blanks automatically given, for instance, a list of keywords that could be blanked and percentage of transformation into blank spaces. However, this restricts the teacher interaction. The student also had to answer the question's side, not being like that very intuitive. Figures 2.4(a) and 2.4(b) present an example of code question style mockup, first using the Fill in the blank - Matching Items and the second the Fill in the blank - Multiple Options Answer.

2.3.3.A Fill in the blank - Open Answer

In the open answer variation of the fill in the blank, the students will have to write the correct answer. They will generally have no hint like provided with the other two variations from having options. This question type concretely is one of the types provided by some of the MOOCs mentioned before [34].

Funabiki et al. [44, 45] evaluate these question types by comparing with the correct ones. This solution is sufficient for their use case since they are only using java reserved words if there is a typo or an incorrect casing; this would not be relevant. Also, since their approach is one word, it is more than enough. However, when we start combining multiple words or allowing the students to create a lambda expression, the evaluation becomes trickier. One possible solution is using edX [38] approach into open answer questions and resort to regex or a list of possible correct solutions, taking or not casing into consideration. The more advanced scenario we try to accomplish with open answers, the more complex solutions we need to provide. Ultimately, recurring to a system like Mooshak [30], so that the solution can be evaluated in the background, and with high correctness and concurrency levels. However, considering these solutions or scenarios also defeats the purpose of having a simplified problem like fill in in the blanks.

2.3.3.B Fill in the blank - Matching Items

Contrary to the open answer variation, matching items present the correct answers that need to be selected. This variation is particularly interesting, especially in a context where multiple options need to be filled. From a list of possible answers, the responder must select the option that fits best for his blank spot [36]. It exists variations where each answer can be used just once or multiple times.

This question type is simple to evaluate at first glance, it would just be required to compare the answer given by the student and comparing it with the correct answer. This question-answer analysis can become more complicated if the question contains multiple blanks that need to be filled. This increased amount of combinations can also result in partial credit. For this, edX [38] is a useful reference, as they throughout the multiple problems they provide have multiple approaches to this problem as exposed in Multiple Selection section.

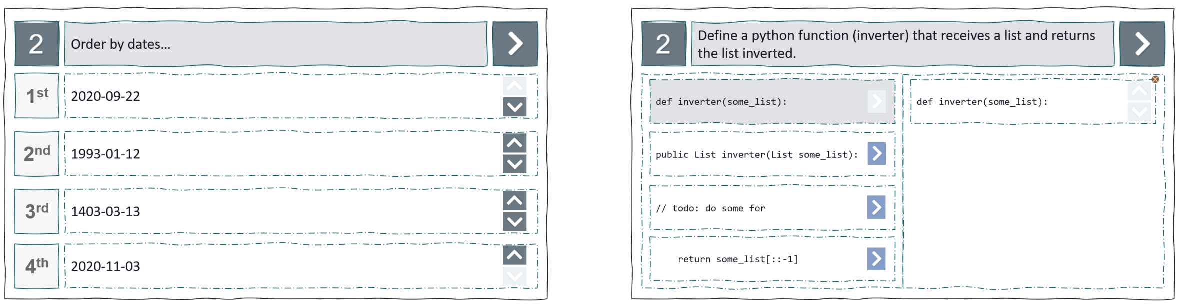
2.3.3.C Fill in the blank - Multiple Options Answer

Finally, multiple options fill in the blank are similar to the matching items one. Each responder will receive a set of options to use to respond to the blank spot. The significant difference is that each set of options will serve only one blank option.

The evaluation of this question type is exactly the same as the Fill in the blank - Matching Items. The only difference is that the examinee will have a select group of options for each blank space whilst in the previous question type they have a set of options that can be used or not in all blanks. So the problem is not actually the evaluation but rather the difference in User Experience (UX)

2.3.4 Sorting Question

Sorting question comprises of having multiple items to a given question that need to be ordered. This question can behave by having all options being part of the solution or only a couple of options requiring the learner to choose the correct options and then order them. A prime example of these questions could be related to ordering steps in a given procedure or events in a given sequence. Figure 2.5(a) and **??** present a mock-up of possible implementations of those questions, on the left we can see one example where the student needs to order all, and on the right an example of both selecting and ordering.


2
Order by dates...
1st
2020-09-22
2nd
1993-01-12
3rd
1403-03-13
4th
2020-11-03
2
Define a python function (inverter) that receives a list and returns the list inverted.

```
def inverter(some_list):
public List inverter(List some_list):
// todo: do some for
    return some_list[::-1]
def inverter(some_list):
```

(a) Order question example **(b)** Select and order question example

Figure 2.5: Mock up order question example

The Parson's Problem [33] is an excellent example of a code question that fits this question type. The Parsons question consists of challenging the students instead of writing code re-order pieces of code to create a working piece of software. The skills required to solve this kind of problem are similar to those required from an entirely written code [46]. Also, they are more straightforward and more reliable when it comes to evaluation. Adding to this a Parsons problem is also a better fit for a quiz, and because it is more efficient than writing or fixing code, it can be an excellent tool for practice [47]. One of the most relevant things about the Parson's Problem [33] is the usage of distractors, this is, a very similar piece of code that is indeed incorrect, and forces students to stop and think.

An evaluation of this kind of problems would consist of evaluating if the pieces were in the correct place. Parson and Haden [33] also considered partial credits, with explanations so that the students could try again and learn from their mistakes. This question is a fascinating code question scenario and easy to evaluate compared to open answer counterparts, the User Interface (UI) on the other hand, might be more complicated than a multiple-choice question.

2.3.5 Open Ended Question

Traditionally exams present multiple open answer questions that allow the student to be exposed to knowledge not being constrained by the multiple options available. When comparing the open-ended questions to the closed-ended questions, as the previous portrayed, is that open-ended questions allow the responder to answer without being influenced [48]. They usually do not appear in quizzes; however, they can be valuable in a questionnaire scenario. Nevertheless, as identified by Chauhan and Goel [34], they exist in almost all MOOC platforms analysed as a question option. As noted by Chauhan and Goel [34], the most common are short text answers and numerical. The numerical answer questions are usually associated with calculation questions, which are especially common in science courses [49].

This question type is very versatile as it can be used to replace almost all others. For instance, if we have an order question, we can easily replace with an open answer question as we can see in the fig. 2.6.

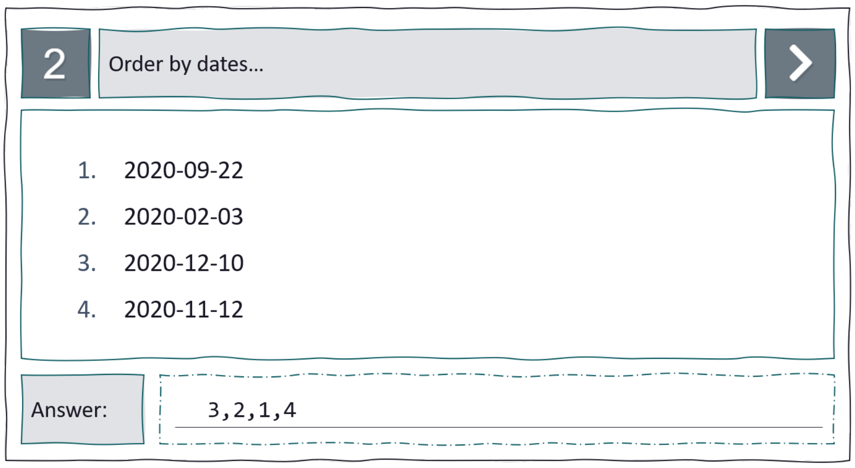


Figure 2.6: Mock up open answer question that could replace open answer

Open-ended questions are straightforward to implement in terms of UI and UX, and even domain implementation used. However, it is more difficult in terms of answer validation, especially when compared to other question types. Firstly in all other question types seen before, we have many constraints in what the answer can be, usually a specific option or a combination of options. In this simple example presented on fig. 2.6 we do not have those constraints easily. The student could insert numbers that do not even exist (for example 5 or -10), the user could use the wrong separators or a combination of other things like inserting letters that we are not expecting. The data validation is a problem, yes, but it can be roughly fixed or controlled, for instance, we could limit the inserted data that could be inputted on the open answer.

Like it was said before, some of the most common uses of open-ended questions in quizzes are either short answers or numerical answers. For the first scenario, the short answers one, edX [38] uses two solutions: a list of correct answers to match or regex pattern. In both of those solutions, it can perform the comparison considering the casing or not of the answer. As for the numerical answer types, they allow once again a list of correct answers to compare, a range or adding an answer with a certain tolerance. The tolerance is particularly interesting as it can try to prevent calculation errors. EdX [38] also has more complex open-answer assessments, but the evaluation is not automatic but rather by peers and teachers. This evaluation process provides a 360 evaluation and tries to ensure some fairness int the process. In a 360 evaluation, there exists a self-evaluation, peers evaluation and teacher evaluation.

This types of open-ended question are the equivalent of an exam question to write a piece of code. As noted in the previous section, tools like CodeRunner [27] and CS Circles [28] are an example of that. The significant difference from the other open questions scenarios is that, contrary to natural language, programming questions can be easily evaluated, even considering students' creativity. We can, for example, focus on the following criteria: does it have any error in the written program (static analysis); does it have any errors in the logic/does it do what is supposed (dynamic analysis). Another ordinary use case for the open-ended question type in programming scenarios is having code snippets that one must understand and answer with the correct output of the snippet execution. Usually, this execution ends in a numeric value. QuizPack [31] and QuizJet [32] are a great example of this use case. In these cases, the correctness of the question cames from the code execution and comparison with the answers. Note that this is only required because they use Parameterized questions(explained below). If the script was static, the approach planned was having the teacher inputting the response.

2.3.6 Parameterized questions

Whilst Open Ended Question are very versatile and can eventually replace any question type with certain limitations as noted. On the other end, parameterised questions are more like a feature that can work together with all other question types. The idea behind a parameterised question is that variations of a given question can be created from a question template which receives parameters.

Figure 2.7 shows an example of the way parameterised questions can be used. In this example, the teacher would set up the default problem as a template. In this template, he would define what could vary, as depicted, in this example, it could be the radius; this means each student would receive a radius that would be comprehended between 5 and 100. Adding to that, the teacher would add a way to obtain the result. This case would be to calculate the area of the circle $A_{rea} = \pi R_{adius}^2$, where the radius is the one attributed to that question instance. This example uses an open answer question, but this could easily be translated into a multiple-choice question.

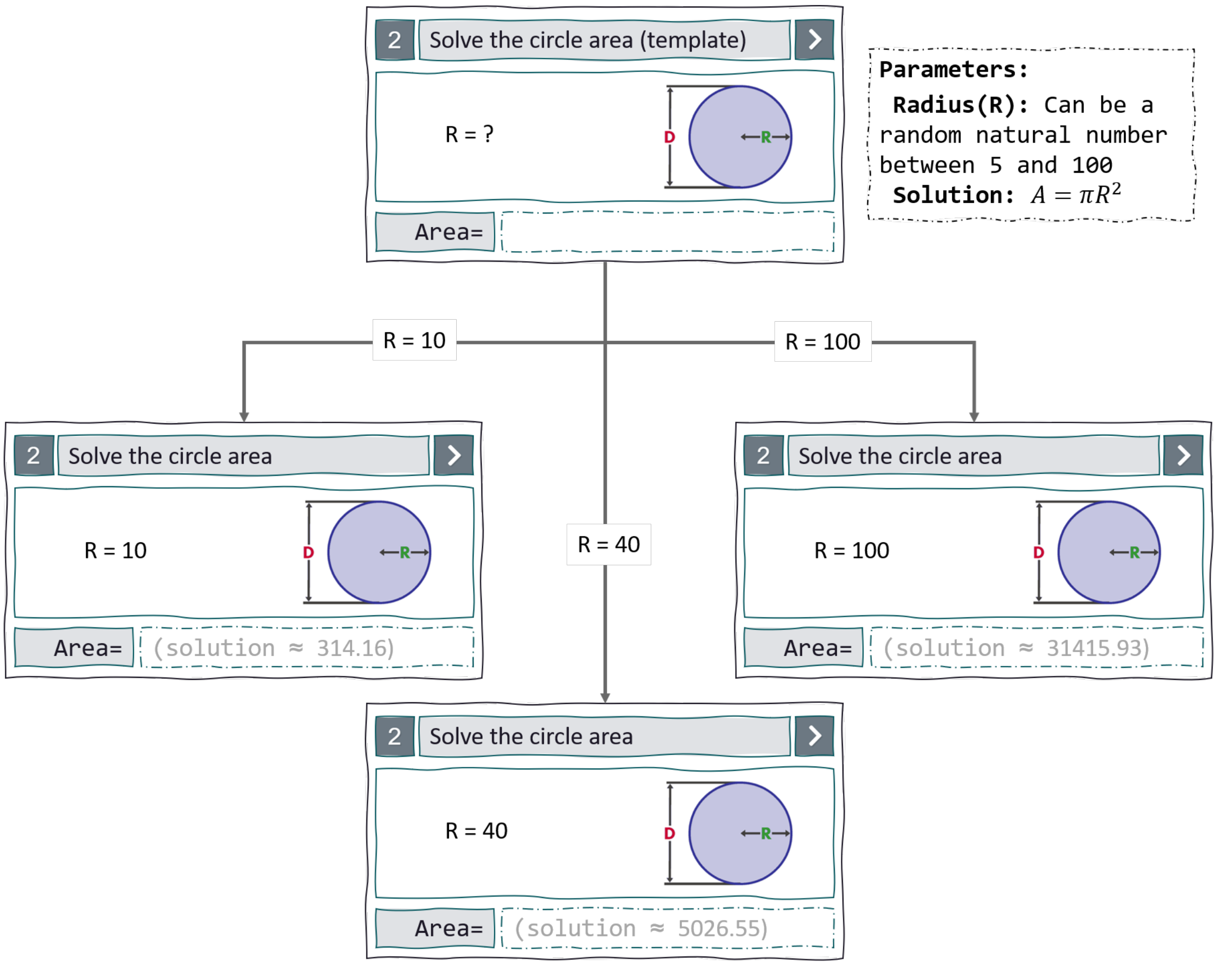


Figure 2.7: Parameterized question example using a mathematical question

Systems like CAPA [50] or WebAssign [51] were some of the pioneers in the exploration of the parametrised questions. However, at the time, not all were automatically evaluated, some solutions required manual intervention, sill it was a significant improvement and with a large acceptance level from the community. For instance, CAPA [50] encourages students to study together, because in the end, each will have to deliver their own answer, but working together is an efficient learning strategy. These and other pioneers of the area demonstrated multiple benefits in exploring this personalised and parameterised approach.

The previous example showed the applications of this question type in a mathematical problem, but another common problem approached with it is programming code questions. We have for instance QuizPack [31] and QuizJet [32], both systems for authoring, delivery and automatic assessment of parameterised quizzes and questions, the first one focusing on the C programming language and the second one on Java programming language. These work in a similar manner, the most relevant part is that for each question a teacher would input a piece of code, on that code they could introduce one variable, that could be replaced by an interval value. The code is delivered to the student, and with the random number calculated in the interval, the code created by the teacher runs on the server code to generate the correct answer for the question validation. Another exciting system add-on that complemented both systems was QuizGuide [52] and JavaGuide [53] both tools that provide an orientation for the student so that they may focus their efforts on their most significant difficulties.

The evaluation challenges with this type of question vary from question type to where it is applied. Nevertheless, one problem is constant, that is finding the correct answer. For example, if one uses a parameterised multiple-choice question, the correction would be only validating if the correct answer is selected. However, it is not trivial defining the right answer. If we are thinking about mathematical problems, then those would be the calculation of the formula. If we are considering code problems, then like QuizPack [31] and QuizJet [32], the better approach would be to execute the code with the new parameters and have that as the correct answer.

2.3.7 Question Types Summary

Besides the question types analysed by Chauhan and Goel [34] and exposed above, platforms like edX [38], for instance, provide an innumerous amount of different question types with multiple possibilities. Regardless of that, this analysis performed on the previous sections paint a clear picture of what is currently done. More than that, it explains the applicabilities with code questions as well as possible challenges. Table 2.1 summarises the information gathered from the multiple sources used above.

Table 2.1: Question types as code question analysis - summary table

Question Type		Exists in Quizzes Tutor?	As code questions	Evaluation Difficulty
Single Selection	True or False	✓	Can be confusing, but useful for code analysis scenarios	Very easy, match option to correct
	Multiple Choice	✓		
Multiple Selection		✗		Easy, match options to correct (complexity might increase with multiple grading scenarios)
Fill in the blanks	Open Answer	✗	Very interesting for question scenario. A good balance between writing code or multiple-choice. Note that all this question types are considerably more difficult to implement than the previous ones.	Hard, multiple solution like regex or accepted list of correct answers can be used (complexity lies in the matching, with the increase of words in a blank space complexity increases).
	Matching Items	✗		Easy, match options to correct (complexity might increase with multiple grading scenarios)
	Multiple Options	✗		Easy, match options to correct (complexity might increase with multiple grading scenarios)
Sorting Question		✗	Parsons Problem	Easy, match options to correct (complexity might increase with multiple grading scenarios)
Open Ended Question		✗	Code challenges	Very hard, require specialized architecture; for each challenge a complex set of tests need to be built

3 Solution Overview

Contents

Quizzes Tutor is a platform in constant evolution, now adding a new programming code question. The current section dissects the solution and steps taken for adding the new code question.

It starts by giving a brief technical overview of the Quizzes Tutor solution, going through both methodologies used as well as technologies. Afterwards dives into the requirements for the new code question and which problem it comes to solve.

Finally, the last two sections leap toward the definitive solution. Firstly, exploring the tackling of the domain transformation to allow for multiple code questions types that in turn, allow for the new code questions. Secondly, it explains the development behind the new code question, exploring the challenges and the decisions taken.

3.1 Background

Quizzes Tutor is a web application for the creation and management of quizzes. It aims to be a tool that students can use to answer both teacher-created quizzes as well as auto-generated quizzes from a poll of questions created by their teachers. It takes both the role of knowledge validation by the teachers as well as a useful self-assessment tool that students can use to improve their learning. Quizzes Tutor is built in a 3-layered architecture:

- Presentation Layer
  - Responsible for displaying information to the user.
  - Technologies used: Vue.js[1] + Typescript[2]; Cypress[3] for unit and integration tests.
- Business Layer
  - Responsible for all business logic.
  - Exposes API to access resources.
  - Service layer to define a facade on top of the domain layer.
  - Domain layer for business logic implementation and database interactions.
  - Access and permission management.
  - Technologies used: Java 11 + Spring-boot; Groovy for unit and integration tests.
- Data Layer
  - Responsible for persisting information.
  - Technologies used: PostgreSQL

[1] https://vuejs.org/
[2] https://www.typescriptlang.org/
[3] https://www.cypress.io/

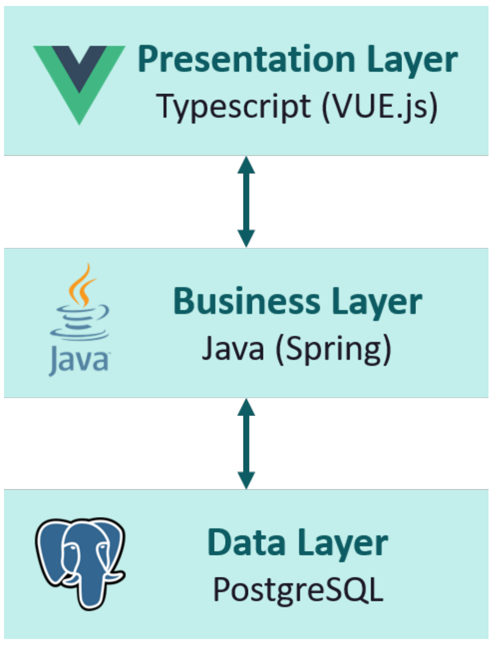


Figure 3.1: 3-Layered Quizzes Tutor Architecture

3.2 Requirements

Quizzes Tutor was never from its origin a dead project. Both students and teachers kept on evolving the tool to make it more robust, to allow for an increased number of parallel users, to incorporate essential features for the functioning of the tool both as a learning and teaching instrument. As referred before, the project only allowed for multiple questions with four options, with only one being correct. Since the beginning of this work, this limitation of having to have four options was removed. Nevertheless, there were still some limitations.

The major challenge that initiated the current work was the creation of programming code questions. The idea was that somehow we could replicate some of the most seen questions on exams and that they could be somehow transformed into a quiz question.

By analysing a set of exams, it was noted that two question types were frequent and relatively simple to be adapted into a quiz question format. The first one consists of giving a snippet of code and finding the ”bug” or the incorrect parts, indicating the line(s) and fixing it. The second question type focused more on completing the code; this is, given a snippet of code the student needed to implement the remaining parts to make it work. This analysis was an interesting result because it would be much simpler to do any of these then to do something like a code interpreter, and even if we where to do that, it would be less relevant or engaging in a quizzing scenario. Nevertheless, this scenario is still interesting for the platform.

However, before starting to understand what kind of data was required, or what would be required to make the programming questions work, one thing became clear: the Quizzes Tutor code needed to be restructured. The Quizzes Tutor application was, as affirmed earlier, very limited in the fact that only supported multiple-choice questions. One possible workaround would have been to try to use the existing questions as a means to ask the code questions, but this would be counter-intuitive, both for the student when answering and for the teacher when creating. Creating a ”question type” would be very relevant for the platform to grow for other requirements. So it was required to a domain transformation to accommodate the question types.

After understanding the domain problem, the programming code question type requirements had to be clear—there where 3 points of view to consider.

Firstly from the teacher standpoint, they would require a way to create and edit these new questions, as they already did with the multiple choices question, with ease of edition using a programming code language-aware editor. It was also relevant that adding a question of each type would be transparent and easy to do when creating new quizzes. Finally, it was critical for the teachers that the problems' evaluation was straightforward and easy to understand. Note that with multiple choices, the response was correct or not, with programming code questions becomes more complex, whichever design is created.

Secondly, from the student standpoint, they needed to have a swimmingly integration with the existing questions and quizzes—all the existing features needed to continue to exist, independent of the question type. Also, the answering of the new code questions needed to be intuitive.

Finally, the standpoint of the developer or maintainer of Quizzes Tutor; for this final standpoint, the code needed to be created according to best practices used across the project. Moreover, it may not have an enormous impact on the overall system both in terms of performance and in terms of the development of new question types. New code questions should be seamlessly created without a considerable overhead from the developer.

Summing it up the requirements became clearer and more refined as the project kept being developed but they summed up to:

- Easy to create code questions.
- Easy to answer code questions.
- Easy to evaluate code questions.
- Seamless integration of code questions with other types of questions.
- Flexible architecture that results in a low development cost of adding new types of question.
- Preserve previous performance levels.

3.3 Multiple Question Types

The first requirement to accomplish was to allow the existence of multiple question types. As explained before, the system was created with a simple architecture and design to accommodate multiple-choice questions, with four options on which one was the correct. Some of these constraints were only code-based. However, allowing multiple question types is a domain problem where an evaluation is required to understand the best approach.

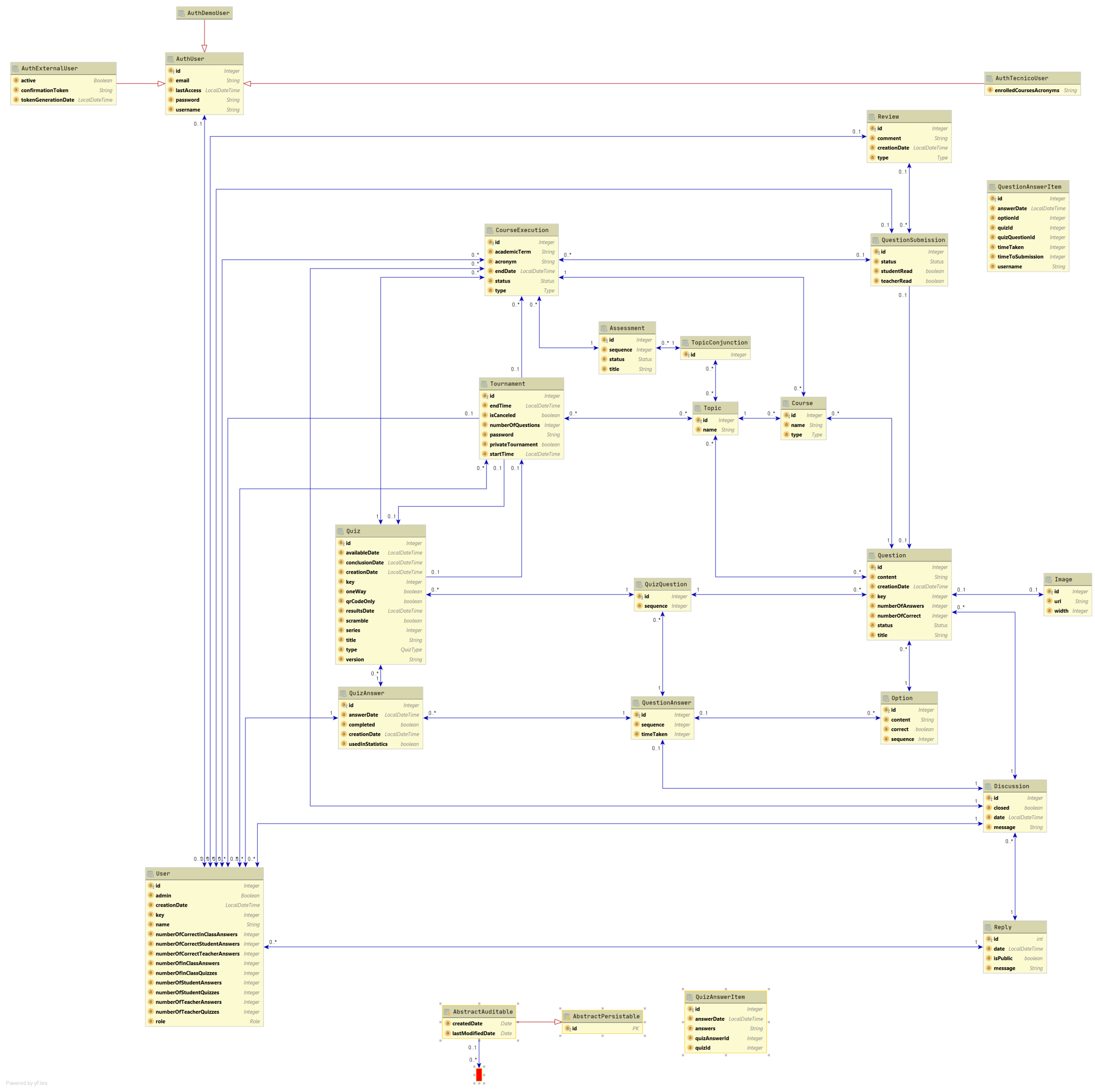


Figure 3.2: Domain model without multiple question types

Figure 3.2 presents a representation of the domain without multiple question types. We can identify multiple entities, and bellow are described some of the more relevant:

- Quiz - Aggregates questions and all the configurations necessary in order to create a quiz.
- User - Represent both the teachers and the students.
- Question - All relevant information for question management.
- QuestionAnswer - All relevant information regarding the answer to a question.
- Course - Represents a specific course

Note that we can identify two main focus areas when thinking about this migration/transformation of our domain: questions and question answers. These are the only ones that will be affected by the existence of multiple question types, for example, quizzes use questions the same happens for quiz answers but should continue to work independent of the question type, they just work with question concept.

The question part of our domain is responsible for holding all the question information needed to manage and have the quizzes' questions. We can classify fields to understand if they should be shared across all question types or specific for a given question type. This classification is an important analysis as it helps understand the domain and where we can draw a line regarding what should be question-specific or question-common fields and properties. Decomposing the domain of the question, we can see the following relevant fields, fig. 3.3 highlights the relevant ones:

- Management Fields - fields that hold the purpose of holding question metadata (e.g. `creationDate` - date of question creation);
- Question Common Fields - fields that are relevant for a question, however, all question types can have them (e.g. `content` - actual question text);
- Question Specific Fields - fields that are only relevant for a given question type (e.g. `options` - option answers, for open-ended questions this would be irrelevant).

Parallelly to the Question concept we have the Question Answer. It is responsible for the tracking of the students' answers to questions. These two concepts are coextending, so for each question, there will be a complementary field in the answers, and this will be even more noticeable with multiple question types. For example, a question has options to choose from, so the question answers must have an option associated with each answer. Furthermore, if a question were open-ended, then the question-answer would require a free text field, for instance. These concepts complement each other domain wise. Like we analysed for questions, it is important to examine question answers to map out the fields that should be shared across all question types or specific for a given question type for the answers. The

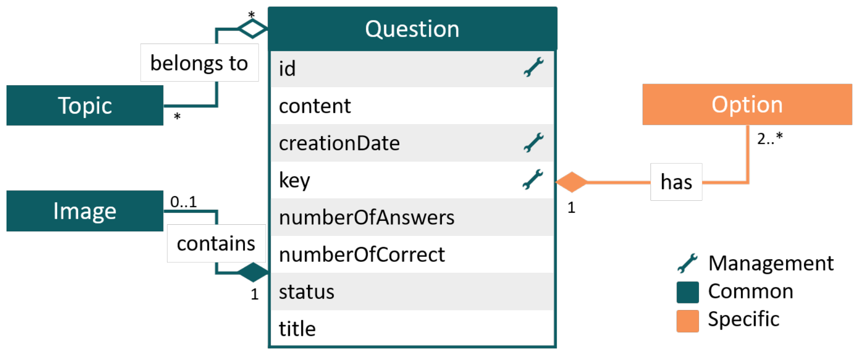


Figure 3.3: Question domain model, highlighting specific fields

decomposition presents a similar separation to the question domain, fig. 3.4 highlights the divisions on the domain of the answer:

- Management Fields - fields that hold the purpose of holding question-answer metadata (e.g. `id` - unique identifier of the answer, used for data management);
- Question Answer Common Fields - fields that are relevant for a question-answer and all question types will have as well (e.g. `timeTaken` - time spent on answering the question);
- Question Answer Specific Fields - fields that are only relevant for a given question type in terms of answering (e.g. `option` - option selected to answer the question, for open-ended questions this would be irrelevant).

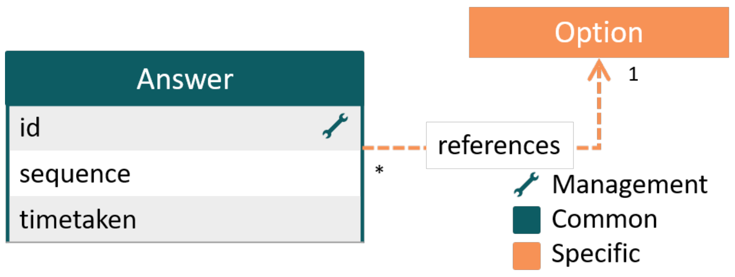
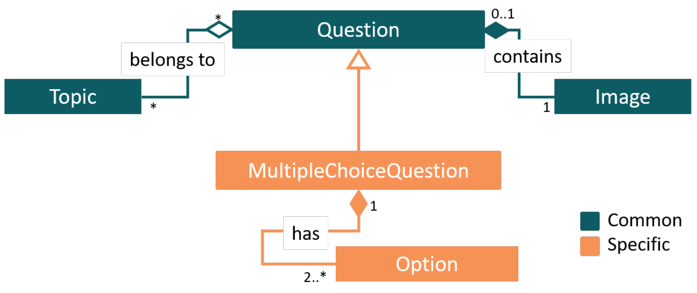


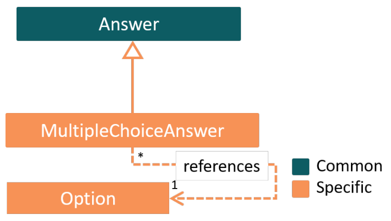
Figure 3.4: Question Answers domain model, highlighting specific fields

Considering these findings and the separation of the fields, it becomes more evident the approach required to modify the domain. It was attempted to perform this transformation recurring to inheritance as a first approach, both on question and question answers side.

This first approach would transform the domain and create specialisations of each question and question answer. For instance, creating `MultipleChoiceQuestion` and `MultipleChoiceQuestionAnswer`, as shown in fig. 3.5(a) and fig. 3.5(b) respectively. This method was a straight forward approach. However, it resulted in some code complexity as it would depend on multiple casts spread throughout the code, which would make the code very error-prone if the correct validation were not made.



(a) Question transformation using inheritance

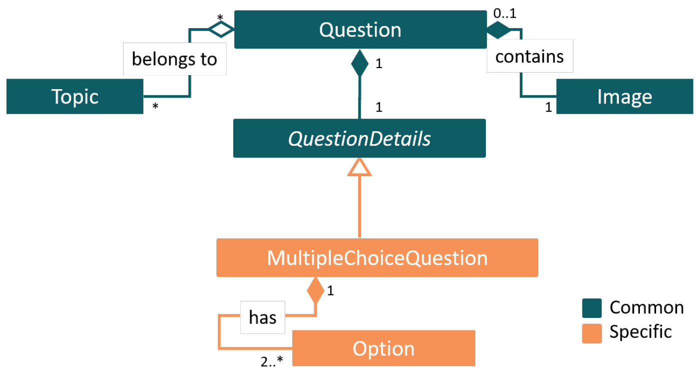


(b) QuestionAnswer transformation using inheritance

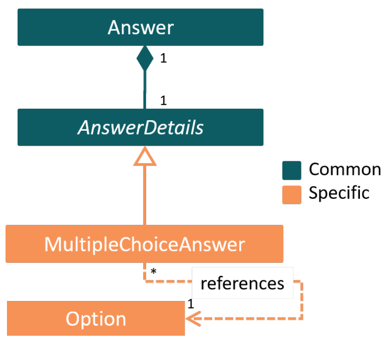
Figure 3.5: Domain transformation of Question and QuestionAnswer using inheritance

Following the evaluation of this first approach, it was decided to proceed in a different direction. The first approach was thinking about adding the Visitor pattern [54] to the existing solution. This solution would solve the casting problem but would create a ”visitor” problem. Using visitor would entail creating specific methods for each visit needed, and the addition of new questions would have to accompany the pattern. This problem is not problematic as we are already using the pattern, so this solution impacts are not large. Nevertheless, one thing became clear, per design, the Question began being responsible for multiple things, so it was decided to split it. Rather than questions being a specialisation of an abstract question, questions would contain question details (same logic for the answers), resulting in favouring composition over inheritance [55]. In this scenario, questions would lose all connections and information specific question type data and all that information would remain the responsibility of question details. As portrayed in the schema in fig. 3.6, we can see the creation of the question details and the specification for each question type, in this case, only multiple choice.

Listing 3.1 presents the code’s differences that in this case, is inserting answer response into the `QuestionAnswer` domain model. The second version becomes much simpler to maintain and avoid errors.



(a) Question transformation using composition



(b) QuestionAnswer transformation using composition

Figure 3.6: Domain transformation of Question and QuestionAnswer using composition

Listing 3.1: Comparing the same snippet functionality before and after the usage of composition and visitor pattern.

```java
public void setResponse(StatementAnswerDto statementAnswerDto) {
    if (statementAnswerDto instanceof MultipleChoiceStatementAnswerDto) {
        MultipleChoiceStatementAnswerDto multipleChoiceStatementAnswerDto = (
            ↪ MultipleChoiceStatementAnswerDto) statementAnswerDto;
        if (multipleChoiceStatementAnswerDto.getOptionId() != null) {
            //validates and sets the answer
        } else {
            //removes the answer
        }
    }
}
```

```java
public void setOption(MultipleChoiceQuestion question,
    ↪ MultipleChoiceStatementAnswerDetailsDto multipleChoiceAnswerDto) {
    if (multipleChoiceAnswerDto.getOptionId() != null) {
        //validates and sets the answer
    } else {
        //removes the answer
    }
}
```

Note that, all these transformations will incur in the creation of more entities. This might impact performance on the Evaluation the solution is put to the test to examine if the solution requirements still hold stable.

Following the main domain transformations performed, it was necessary to conclude all the services, Data Transfer Object (DTO) and tests that depended on the domain to work. All these transformations rely heavily on abstract classes that serve as templates to the specific implementations to be done. Section 3.4.3.B describes both the exiting abstractions in more detail and explains the approach taken when creating the new code question.

Testing is an essential process in the development of any product. Having a good battery of tests was crucial during the migration and will continue to prove useful during the platform's further developments, given the fact that the way to create multiple choice questions was altered all tests needed to be updated.

However, during this stage, Frontend (FE) code was mostly unaffected. DTOs are simple objects that usually do not contain any business logic. The only logic present is storage, retrieval, serialisation and deserialisation of its data. The serialisation and deserialisation are used to transfer information over the web. That being said, the DTOs' serialisation and deserialisation mechanisms were updated on all relevant classes with a new field, the `type`, which maps to a specific question type. If no info is present, this field's default value is `multiple_choice`. Listing 3.2 displays the setup required to achieve this serialisation and deserialisation transformations. FE's code was updated to support new DTOs. Some other less relevant changes to FE were done to improve its code and prepare it for a more modular structure used afterwards to insert the new question types (see section 3.4.4). For example, one of the changes was the addition of `QuestionHelpers` that manages the deserialisation on the FE side. Listing 3.3 presents some of the resulting DTO transformations, in this case, referring to the question model. We can see on the left the old model and on the right the new one. The major difference is the addition of the `questionDetailsDto` and respective `type`.

Listing 3.2: Snippet of code that demonstrates the serialisation management in DTOs

```java
//...

@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        defaultImpl = MultipleChoiceQuestionDto.class,
        property = "type")
@JsonSubTypes({
        @JsonSubTypes.Type(value = MultipleChoiceQuestionDto.class, name =
    MULTIPLE_CHOICE_QUESTION),
        @JsonSubTypes.Type(value = CodeFillInQuestionDto.class, name =
    CODE_FILL_IN_QUESTION),
})
public abstract class QuestionDetailsDto implements Serializable, Updator {

    public abstract QuestionDetails getQuestionDetails(Question question);
}
```

Listing 3.3: Comparison between DTOs serialisation of the question model.

```
{
[...]
  "title": "Sample Question",
  "content": "Example Question",
[...]
  "options": [
    {
      "id": 27865,
      "sequence": 0,
      "correct": true,
      "content": "Option 1"
    },
     [...]
  ]
}
```

```
{
[...]
  "title": "Sample Question",
  "content": "Example Question",
[...]
  "questionDetailsDto": {
    "type": "multiple_choice",
    "options": [
      {
        "id": 27865,
        "sequence": 0,
        "correct": true,
        "content": "Option 1"
      },
     [...]
    ]
  }
}
```

Finally, with the domain changes, it was also necessary to consider the database data's preservation. As such, and as part of this task, it was created migration scripts that would ensure that all old database data would be preserved in the new architecture.Figure 3.7 consists of the final state of the domain with the new multiple question types architecture.

These changes summed up toward a transmutation of the code to provide a flexible and simple to use implementation of question types. These transformations were the initial alignments into creating a flexible architecture that results in a low development cost of adding new types of question. With the development of Code Question Types this requirement is concretised.

3.4 Code Question Types

Following our domain model's transformation, we were ready to introduce programming code questions that would reflect part of the exercises done in exams and class. Revisiting the requirements, there needs to be a good UI — the new UI needed to accommodate both an intuitive and straightforward interface for the teachers and students.

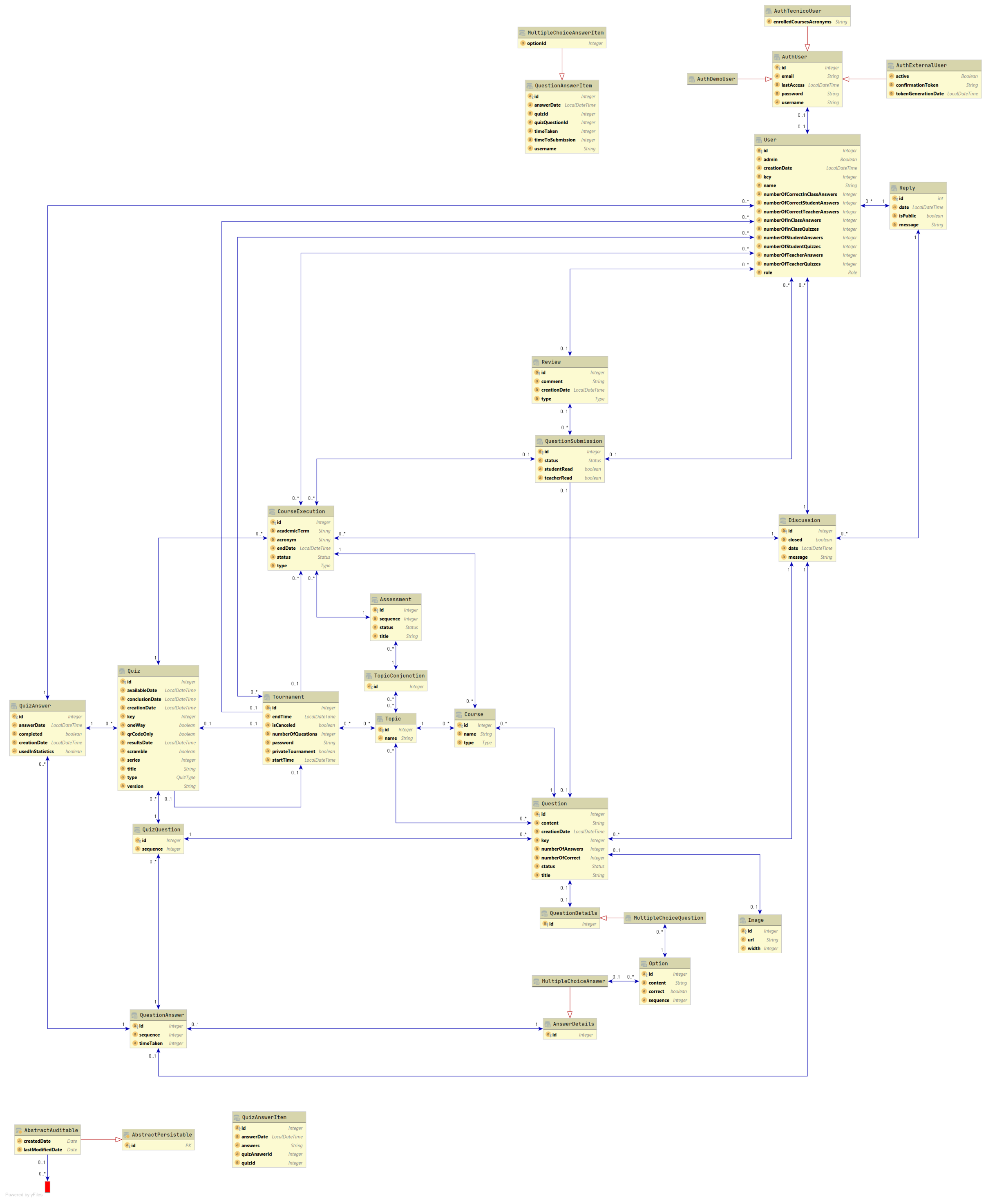
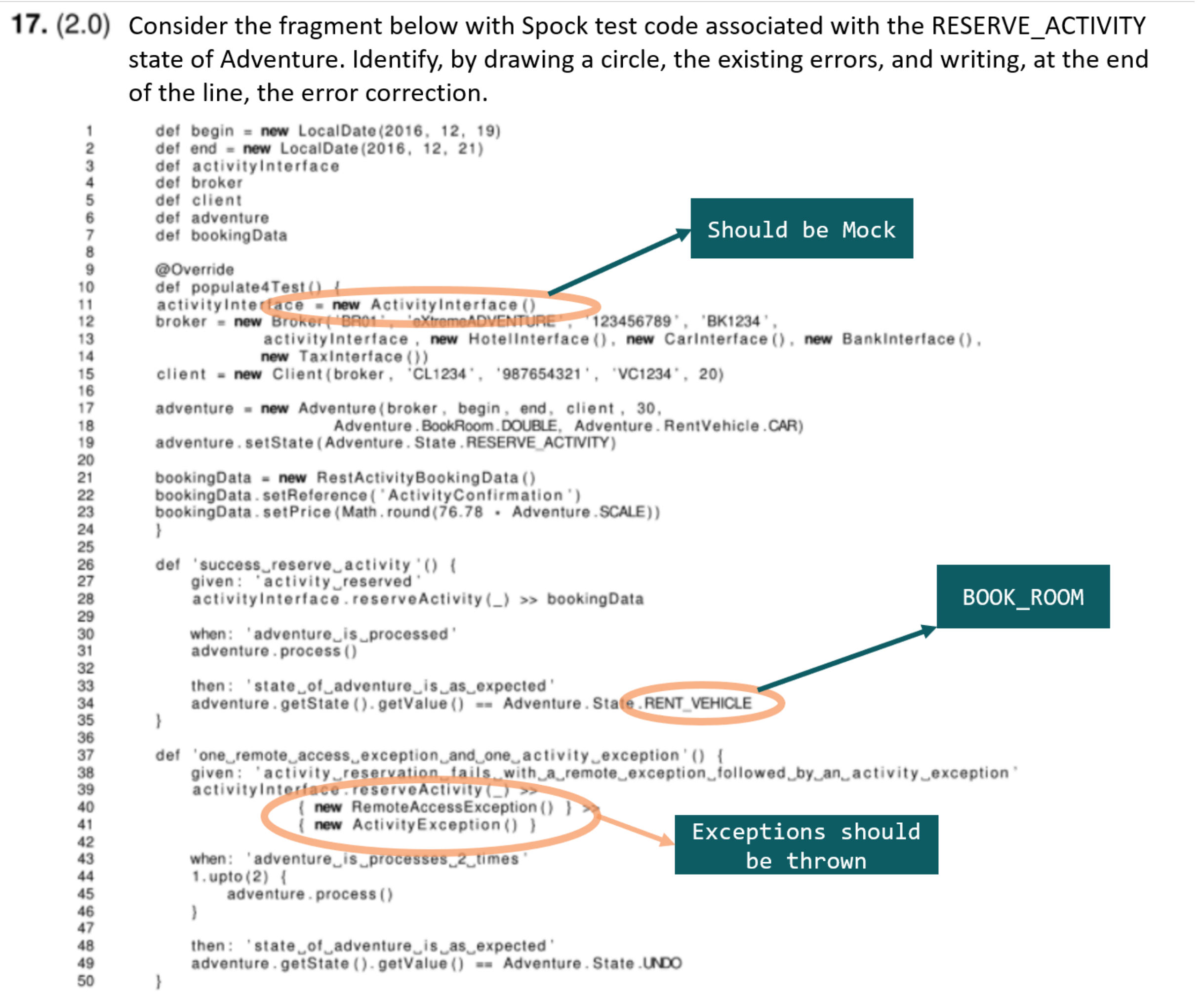


Figure 3.7: Domain model with multiple question types

3.4.1 Understanding the exam code questions

The initial concern was to focus on the student side of the question. That UI is very relevant to understand as well what kind of data structure would be required for constructing code questions. The FE code uses Vue.js, so it requires a solution that takes most of the framework. The first step was to try to create an interface similar to what it would exist on written exams. Figure 3.8 presents an example of the code belonging to an exam question. Here the student would need to identify which lines were problematic and also present a solution for them. These are the kind of questions that would require said conversion to the quiz question counterpart.



17. (2.0) Consider the fragment below with Spock test code associated with the RESERVE_ACTIVITY state of Adventure. Identify, by drawing a circle, the existing errors, and writing, at the end of the line, the error correction.

```
def begin = new LocalDate(2016, 12, 19)
def end = new LocalDate(2016, 12, 21)
def activityInterface
def broker
def client
def adventure
def bookingData

@Override
def populate4Test() {
activityInterface = new ActivityInterface()
broker = new Broker('BR01', 'eXtremeADVENTURE', '123456789', 'BK1234',
               activityInterface, new HotelInterface(), new CarInterface(), new BankInterface(),
               new TaxInterface())
client = new Client(broker, 'CL1234', '987654321', 'VC1234', 20)

adventure = new Adventure(broker, begin, end, client, 30,
                    Adventure.BookRoom.DOUBLE, Adventure.RentVehicle.CAR)
adventure.setState(Adventure.State.RESERVE_ACTIVITY)

bookingData = new RestActivityBookingData()
bookingData.setReference('ActivityConfirmation')
bookingData.setPrice(Math.round(76.78 * Adventure.SCALE))
}

def 'success␣reserve␣activity'() {
    given: 'activity␣reserved'
    activityInterface.reserveActivity(_) >> bookingData

    when: 'adventure␣is␣processed'
    adventure.process()

    then: 'state␣of␣adventure␣is␣as␣expected'
    adventure.getState().getValue() == Adventure.State.RENT_VEHICLE
}

def 'one␣remote␣access␣exception␣and␣one␣activity␣exception'() {
    given: 'activity␣reservation␣fails␣with␣a␣remote␣exception␣followed␣by␣an␣activity␣exception'
    activityInterface.reserveActivity(_) >>
              { new RemoteAccessException() } >>
              { new ActivityException() }

    when: 'adventure␣is␣processes␣2␣times'
    1.upto(2) {
        adventure.process()
    }

    then: 'state␣of␣adventure␣is␣as␣expected'
    adventure.getState().getValue() == Adventure.State.UNDO
}
```

Figure 3.8: Sample exam question, with answers

Before starting to think technically on the solution to be implemented in Quizzes Tutor, the first thing is to identify and understand some of these types of questions' high-level possibilities. It is essential to understand what would be relevant for the quiz scenario and how it reflects what teachers are doing in the classes exams. Conceptually speaking the enumeration below, presents some question possibilities that could mimic the different exam questions. The enumeration below is ordered by complexity and describes both the questions as well as their difficulties.

1. Question - Find bug per line
    - Description: Given a snippet of code identity, the lines where problems might occur. This question would just focus on identifying the problem, a more complex interaction like the exams, where the problem is corrected, can with automatic evaluation, be accomplished in question type 4.
    - Evaluation: Easily accomplished by comparing if the selected lines match the expected correct lines.
    - Scoring:
        - All correct give full marks or something incorrect gives nothing.
        - Partial punctuation, each correct line gives a point, each incorrect removes or does not give a point.
2. Question - Fill in the blanks (Multiple Choice)
    - Description: A quiz question with a code snippet where students use a dropdown to select the correct piece of code that corrects the snippet.
    - Evaluation: Easily accomplished by comparing if the selected option for each dropdown is the correct one.
    - Scoring:
        - All correct give full marks or something incorrect gives nothing.
        - Partial punctuation, each correct option gives a point, each incorrect removes or does not give a point.
3. Question - Fill in the blanks (Open Answer)
    - Description: A quiz question similar to the previous one, the main difference is that here evaluated person will have to write the code instead of selecting it from a dropdown. The code to be written could be a simple word or a full line, should not be more than that.
    - Evaluation: Here, the situation is more complicated than the previous one. Can be as simple as a comparison (Expected ”int” got ”int”), it can be a regex comparison, or in more complex scenarios would be required to perform full code analysis, with dynamic and static analysis of the code. This last option makes this question type more complex and should not be considered as the question types 4 or 5 are more interesting for that scenarios.

- Scoring:
    - All correct give full marks or something incorrect gives nothing.
    - Partial punctuation, each correct option gives a point, each incorrect removes or does not give a point.
    - Might be limited to the correction method.

4. Question - Code ”review”

    - Description: In a code review question takes inspiration in the exam questions where the students need to identify problems in code and fix it. The idea of this question is being like question 1 where student identify the problem per line, but here also need to fix it by changing the code.
    - Evaluation: To evaluate this question, there are two levels: the first and simpler one is if the identified lines are correct; the second is if the correction done does what is expected. To perform the second evaluation, a specialised infrastructure must run the code and test it in a sandbox environment. It will also be a significantly more complex evaluation than the previous since a battery of tests needs to be created for each question.
    - Scoring:
        - Ideally there would be a component for the identification and another for the correction in the scoring.
        - The identification would be similar to question type 1
        - The correction part would be a pondered grade of the automated tests

5. Question - Code challenge

    - Description: Code challenge would be similar to the previous question (4), with the difference that the examinee would need to develop the code fully from a problem statement.
    - Evaluation: Uses a specialised infrastructure to run the code and test it in a sandbox environment. Each question requires a battery of tests to run for each submission.
    - Scoring:
        - The scoring depends on the number of existing tests, similarly to the previous question type (4).

This evaluation and enumeration obtained from analysing the exam questions and mapping them into quiz questions are very important in defining the next steps. After analysing it, it was decided to explore the first two questions in order to understand the actual possibilities.

3.4.2 Sample implementation of exam code questions

Quizzes Tutor had no previous code questions, so before starting to undertake the final solution, it was really relevant to explore the questions identified in the previous section (see section 3.4.1).

To ensure that creating and answering programming questions is intuitive, the solution needs to support many of the things code editors support. Some of the expectations where:

- Highlighting support
- Linting support
- Multiple languages support

The first challenge is to have code in the browser in a recognisable manner. It is expected to find code well indented, with the proper keywords identified similar to what we would find in a code editor. To solve this problem, we turned to a JavaScript-based library. CodeMirror [29]. CodeMirror is a versatile and easy to use library to create programming code boxes in the browser. It has a rich programming API that allowed for an interesting interaction with the code. Another interesting find was the port of CodeMirror to Vue [4], which simplifies the usage with the frontend framework. With one of the most important tools of the programming code questions implementation chosen, we began implementing the exam code questions into a sample code project. This experiment allowed a better validation of the selected questions and tools before integrating with the final solution. Figures 3.9(a) and 3.9(b) expose the first attempts to understand how the students would see the questions to answer and what would be required domain wise.

Besides the experiment with the actual code questions, CodeMirror [29] was also put to the test to understand the possibilities and limitations. After said analysis, it was clear that CodeMirror was a versatile tool that could be used in the FE to present an excellent UI to its users.

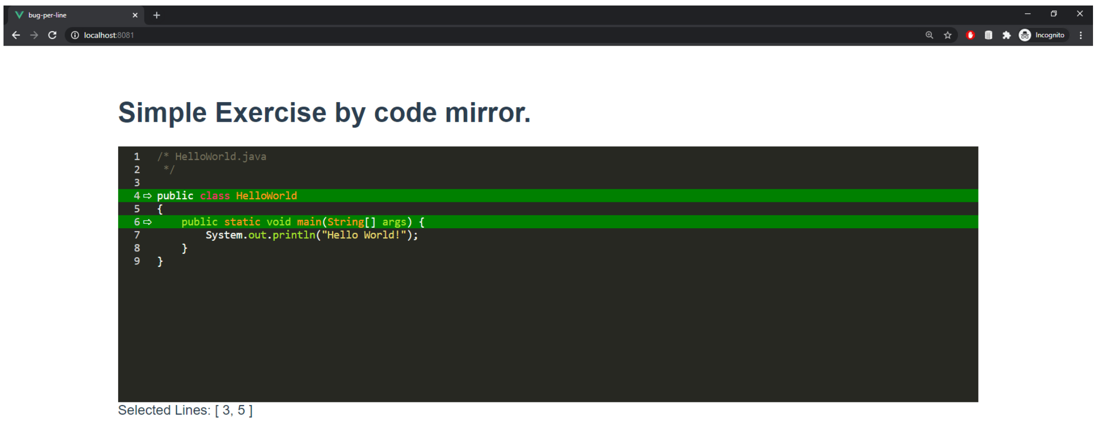
After experimenting and creating some samples, it was decided that the Code Fill In would be the most interesting for the first implementation of code questions. At this phase, we required a clear understanding of how these changes would impact the domain, that was now ready to receive multiple question types.

3.4.3 Backend changes to support code questions

With a good understanding of the code fill in question type challenges and possibilities, it is possible to start performing the code changes, starting with the Backend (BE).

Starting with the domain changes, the following sections will give a high-level overview of the work performed to ensure the addition of the new programming code questions. After having the domain

[4]https://github.com/surmon-china/vue-codemirror

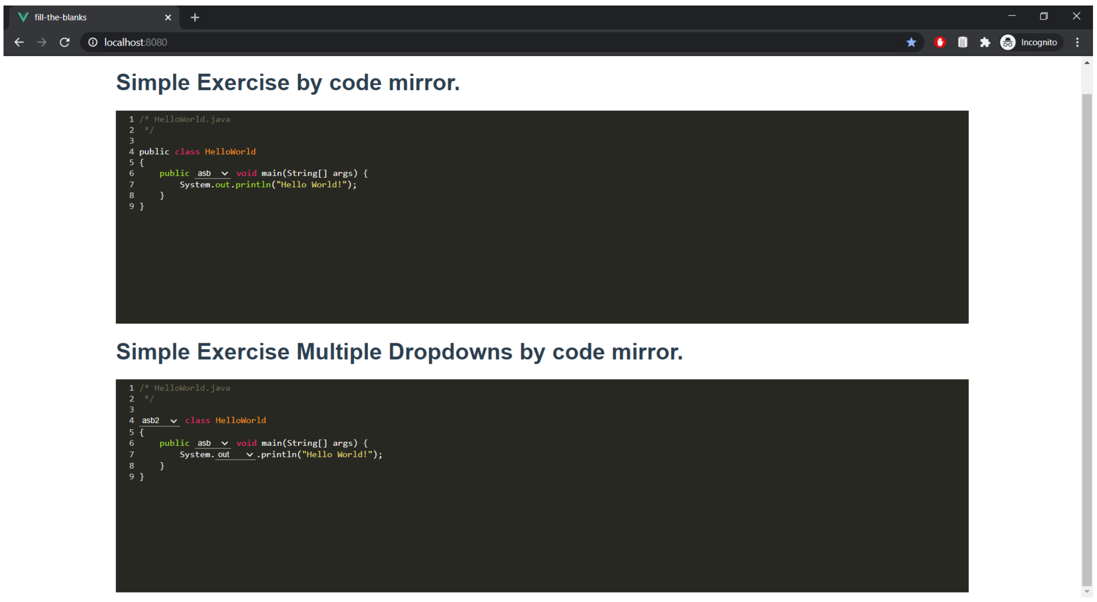


Simple Exercise by code mirror.

```
1    /* HelloWorld.java
2     */
3
4 ⇨  public class HelloWorld
5    {
6 ⇨      public static void main(String[] args) {
7            System.out.println("Hello World!");
8        }
9    }
```

Selected Lines: [3, 5]

(a) Select lines sample question



Simple Exercise by code mirror.

```
1 /* HelloWorld.java
2  */
3
4 public class HelloWorld
5 {
6     public  asb  ⌄  void main(String[] args) {
7         System.out.println("Hello World!");
8     }
9 }
```

Simple Exercise Multiple Dropdowns by code mirror.

```
1 /* HelloWorld.java
2  */
3
4  asb2  ⌄  class HelloWorld
5 {
6     public  asb  ⌄  void main(String[] args) {
7         System. out    ⌄ .println("Hello World!");
8     }
9 }
```

(b) Code fill in sample question

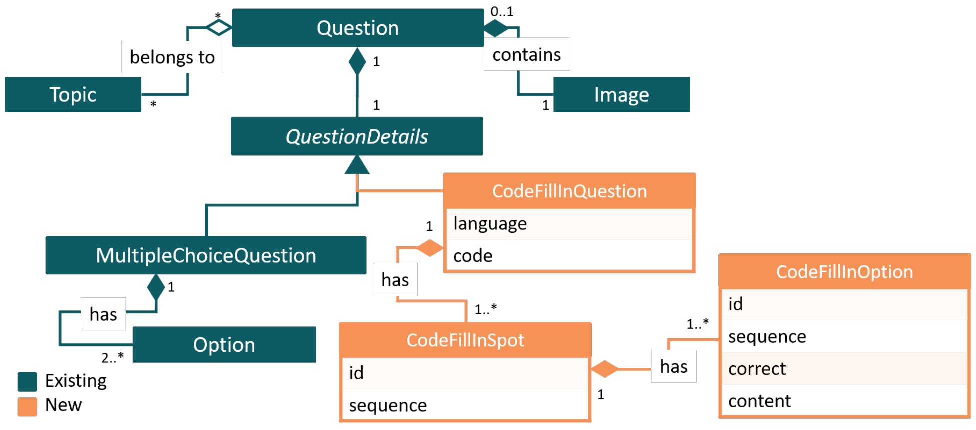
Figure 3.9: First attempts of using CodeMirror to implement question answer visualization

changes comprehended, it dives into the alterations that were done to the remainder of the backend code.

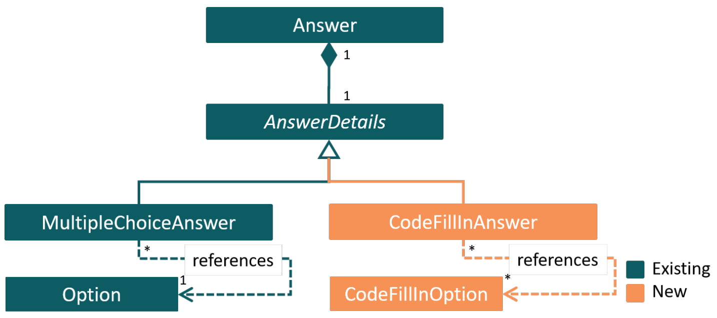
Finally, this section concludes with a summary of all necessary backend changes to include new code questions.

3.4.3.A Domain changes to include code questions

Similarly to what happened with the multiple-choice questions, it is clear that the programming questions with the code fill-in have some properties that are just theirs. For example, these code questions will require a programming language, a code with slots to be filled, the options to said slots. Figure 3.10 portraits the domain transformations done to allow for these code questions. The available languages will be managed on the FE and BE side given that in order to implement new programming languages, we will need to update FE references but also ensure that the BE will support it. This duplication is advantageous so that we, in the future, can incorporate even more complex code questions, like the ones listed in the Understanding the exam code questions sub-section.



(a) New CodeFillInQuestion



(b) New CodeFillInAnswer

Figure 3.10: Question and QuestionAnswer domain model, after the introduction code fill in question

Besides these significant and more noticeable changes to the domain, it was also necessary to include a new code fill in `QuestionAnswerItem`. This domain change allows this question type to have its information logged and enables the system to perform better under large loads.

At this point, with all domain changes identified, it was possible to begin the backend side implementation.

3.4.3.B New backend classes and other changes

The backend side required new domain classes and respective DTO, besides that, it is also indispensable to implement all the appropriate methods to handle each new domain class. The service layer mostly works with the generic questions, hence not being required significant changes on that level. The only other implementation that needs to be done is the export functionality of the Quizzes Tutor application, which allows the data to be exported to multiple formats. To allow for a flexible code infrastructure, the code base relies heavily on the Visitor pattern [54]. This pattern allows for an excellent flexible way of exporting our Questions into the multiple formats required by the application.

Figure 3.11 displays all the changes done organised around the Java packages. As we can see, there are three main packages affected: question, answer and statement. Statement's package consists of classes surrounding the answering of quizzes by the students. The statements' domain holds the items that are a lightweight manner of temporarily keeping the answers before saving and validating its correctness with the answers package. Question's package holds all the relevant classes to manage the actual questions. Finally, the answer's package contains all the information and business logic related to the questions' answers. Although not present in the schema, inside the question package, there are two additional changes. The first that actually should be the first one done is adding the new question type to the `QuestionType` class. The second is the update of the `Updator` class, a visitor interface used to propagate updates.

Observe that all DTO parents are updated; this happens because it is necessary to include the serialisation information for the correct question type and question DTO. Heed that the schema does not consider fixes that were done to previous developments, or tests that were created.

3.4.3.C Backend changes required to add a new question

After finishing the backend transformation, it becomes even clearer the changes required to add new question types to the backend side. Figure 3.12 presents a schema of the developments required to update the backend code. Note that in the schema, the DTOs and remainder classes only show inheritance properties. On the previous schema, this did not happen. This difference comes from the fact that not all of those changes might be required. However, all in the schema of fig. 3.12 are indeed

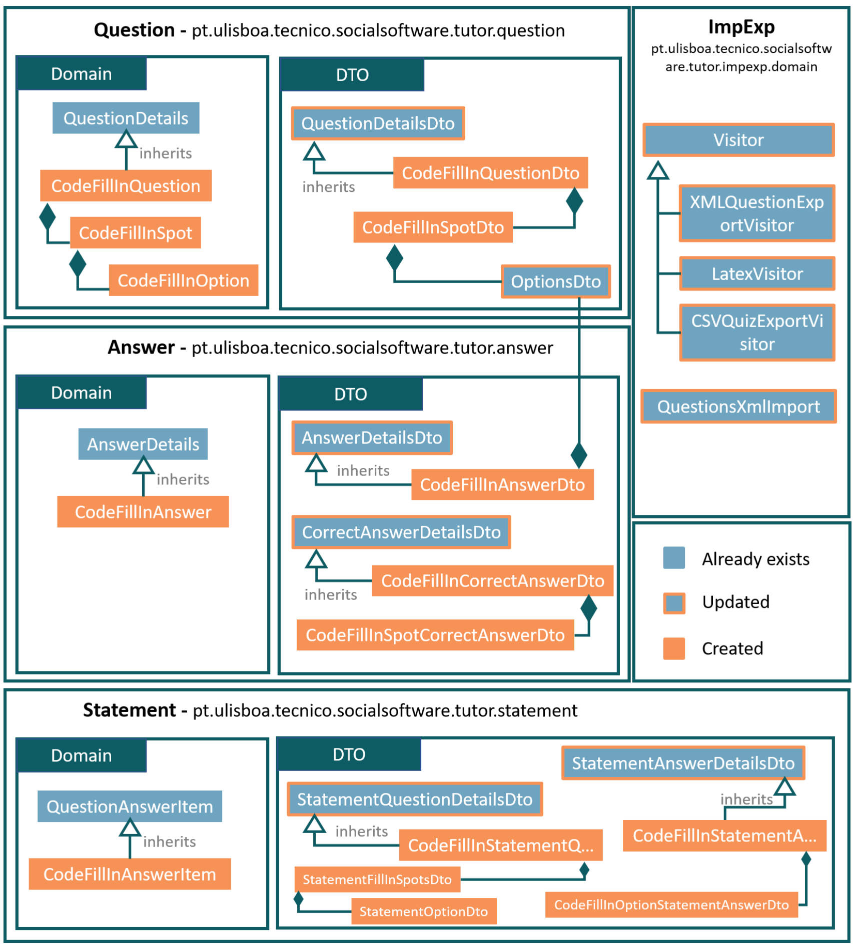


Figure 3.11: Backend changes required to introduce the new Fill In Code question

needed. All other classes just complement the implementation and ensure that the business logic works correctly.

Each specific domain needs to be evaluated independently, but as the schema suggests the changes to introduce a new code question are quite mechanic and simple. The complexity depends only on the question itself, and which other classes might require. Once again, this schema does not present any tests which are clearly necessary. Also, as noted before, inside the question package, there are two additional changes. The first that actually should be the first one done is adding the new question type to the `QuestionType` class, in this case would be "`generic_example`". The second is the update of the `Updator` class, a visitor interface used to propagate updates.

3.4.4 Frontend Changes

Considering the backend tested and finished, frontend was up next. As mentioned before, CodeMirror was a vital part of this solution as it provided all the code editor functionalities used. As previously

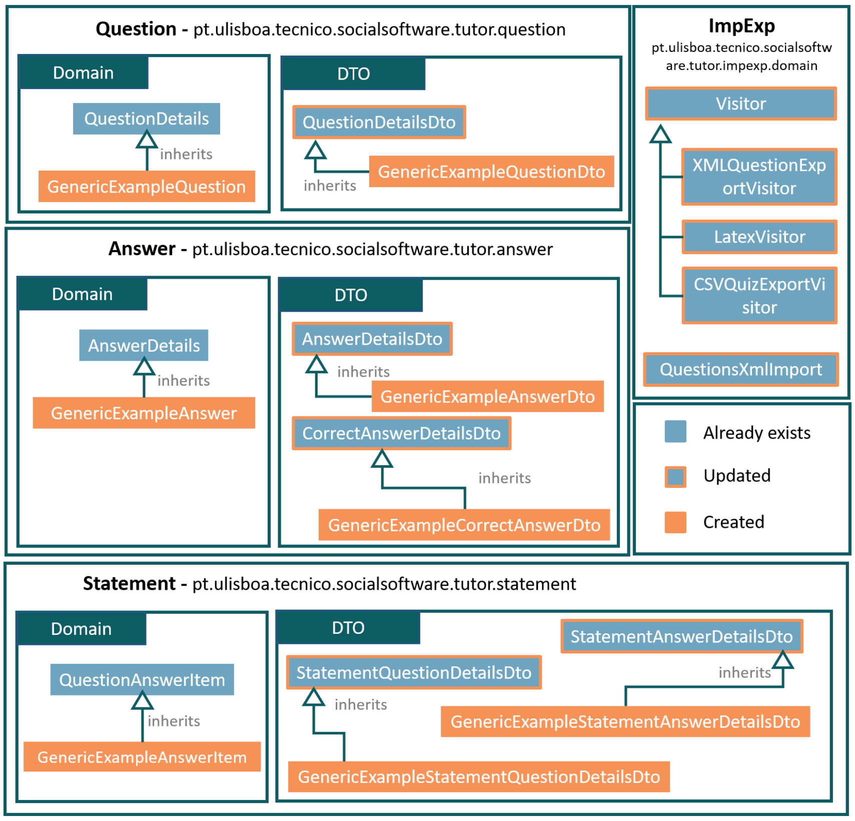

Question - pt.ulisboa.tecnico.socialsoftware.tutor.question
Domain
QuestionDetails
inherits
GenericExampleQuestion
DTO
QuestionDetailsDto
inherits
GenericExampleQuestionDto
ImpExp
pt.ulisboa.tecnico.socialsoftware.tutor.impexp.domain
Visitor
XMLQuestionExportVisitor
LatexVisitor
CSVQuizExportVisitor
QuestionsXmlImport
Answer - pt.ulisboa.tecnico.socialsoftware.tutor.answer
Domain
AnswerDetails
inherits
GenericExampleAnswer
DTO
AnswerDetailsDto
inherits
GenericExampleAnswerDto
CorrectAnswerDetailsDto
inherits
GenericExampleCorrectAnswerDto
Already exists
Updated
Created
Statement - pt.ulisboa.tecnico.socialsoftware.tutor.statement
Domain
QuestionAnswerItem
inherits
GenericExampleAnswerItem
DTO
StatementAnswerDetailsDto
inherits
GenericExampleStatementAnswerDetailsDto
StatementQuestionDetailsDto
inherits
GenericExampleStatementQuestionDetailsDto

Figure 3.12: Backend changes required to introduce the new generic question (named: GenericExample)

stated, the frontend required the following interfaces: teachers' interface to manage questions, teachers' interface to manage quizzes, and students' interface to answer and validate it.

Before starting tackling the actual UI changes in the FE, it is relevant to prepare it to be updated and create the essential FE components and models.

3.4.4.A Frontend structure update for new question

After the BE being created, it becomes clear the models that will be required to be created on the FE side. Following the DTOs created on the BE side, as seen in table 3.1, we can start mapping them into Typescript. We can see that for each DTO in the BE there is on in the FE. Note that for simplicity, not all mappings are described in the table below.

Besides the models, two other structural changes that should be tackled before adding a new code question. Firstly one should update the `QuestionHelpers` to prepare the models to be correctly de-serialised from the FE side. Secondly, setup, like done for the multiple-choice questions section 3.3, components to be implemented with a shared UI across the platform. The components required are:

- Question Visualisation (`CodeFillInView.vue`) - Responsible for presenting a question in a read-only mode. Is also used to present answers given by the students to the teachers with a simpler

Table 3.1: DTOs to Model Mapping

Backend DTOs	Frontend Models
CodeFillInAnswerDto	mngmt/questions/CodeFillInAnswerDetails.ts
CodeFillInQuestionDto	mngmt/questions/CodeFillInQuestionDetails.ts
CodeFillInStatementAnswerDetailsDto	stmt/questions/CodeFillInStatementAnswerDetails.ts
CodeFillInCorrectAnswerDto	stmt/questions/CodeFillInStatementCorrectAnswerDetails.ts
CodeFillInStatementQuestionDetailsDto	stmt/questions/CodeFillInStatementQuestionDetails.ts

UI

- Question Creation and Edition (`CodeFillInCreate.vue`) - Responsible for creating or editing a question. This type of component will then be used for question submission and teacher question management (see section 3.4.4.B)
- Question Answer (`CodeFillInAnswer.vue`) - Used by the students to answer the actual question.
- Question Answer Result (`CodeFillInAnswerResult.vue`) - Used by the students to validate the answer.

If the components are simple enough, they can be condensed into less, used across all views. For instance, the multiple-choice question shares the answer and answer result components.

With the FE structure setup, we can continue implementing the specific UIs.

3.4.4.B Teachers' interface to manage questions

The first interface developed was the teachers' interface to manage questions. On this interface, they were already able to create and manage all the multiple-choice questions. It was required to extend the creation to allow for multiple types. This extension was done by adding a select-box right on the top of the `dialog`. We can see this on the pictures fig. 3.13 (before the multiple types of questions) and after fig. 3.14. Note that the UI differs only on the bottom part of the `dialog`. This detail was a concern that existed throughout the development of the FE that focused on creating a UI as shareable as possible across all question types. Now specifically for the question management of the code fill-in question, it was required the development of two different UIs, one for the creation and the other for visualisation.

Firstly the creation/edit one, fig. 3.15(a) depicts the new UI. On (1), we can see the language selection box; this will determine the programming language we want our code to be. On (2), we have our code editor, which will be highlighted with the correct programming language. (3) shows the button that allows fill in slots to be created. Basically, one selects the text they want to convert in a dropdown, and a new slot is created immediately, having as correct answer the text that was already there. Finally, (4) presents the answer spots which can be added incorrect options to give to our learners to test them.

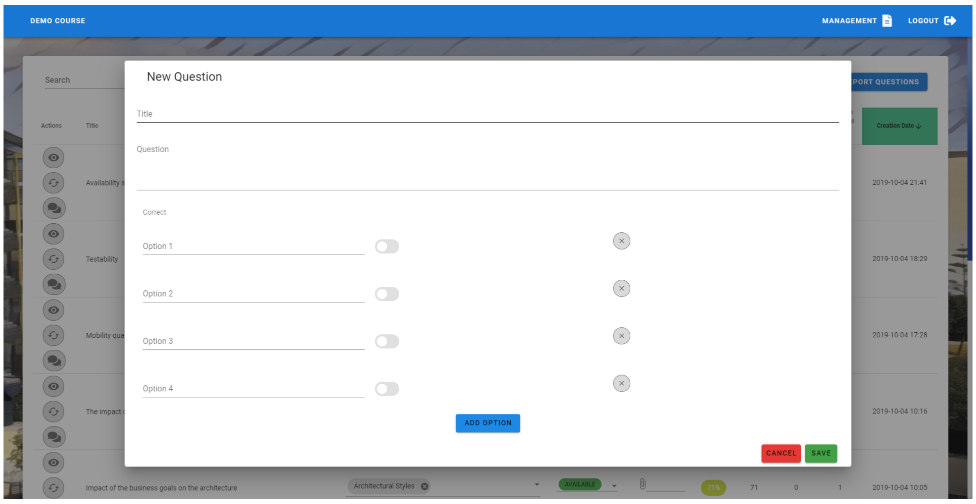
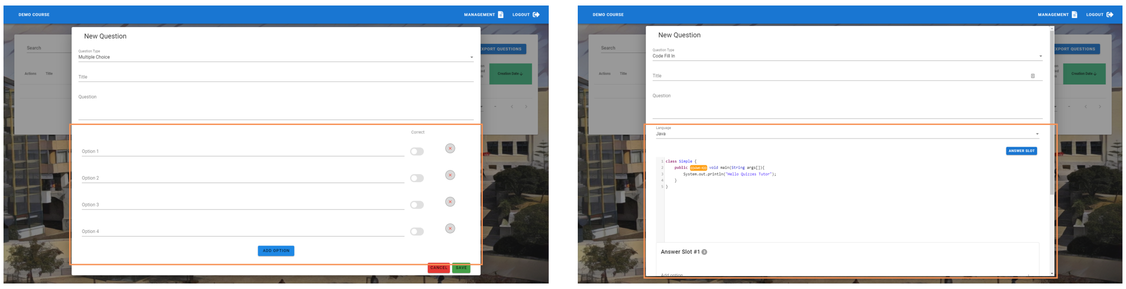


Figure 3.13: Old code question creation UI

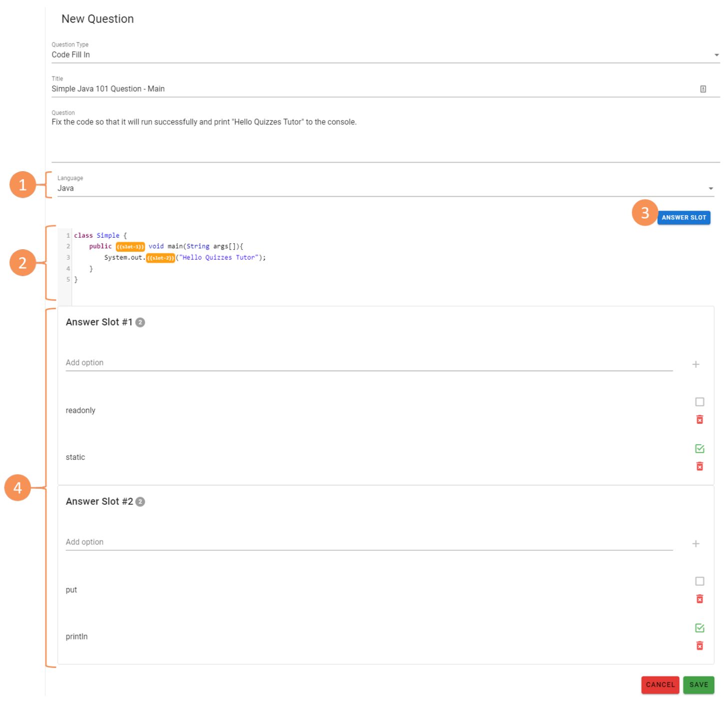


```
class Simple {
    public {{slot-1}} void main(String args[]){
        System.out.println("Hello Quizzes Tutor");
    }
}
```

(a) Multiple Choice Question Creation Example **(b)** Code Fill In Question Creation Example

Figure 3.14: New code question creation UI taking multiple question types into consideration

Note that fig. 3.15(b) shows the edit UI as it can be seen it is exactly the same as the create one with the distinction that some options are ”locked” and cannot be changed.



New Question

Question Type: Code Fill In

Title: Simple Java 101 Question - Main

Question: Fix the code so that it will run successfully and print "Hello Quizzes Tutor" to the console.

Language: Java

```java
class Simple {
    public {{slot-1}} void main(String args[]){
        System.out.{{slot-2}}("Hello Quizzes Tutor");
    }
}
```

Answer Slot #1

Add option

readonly

static

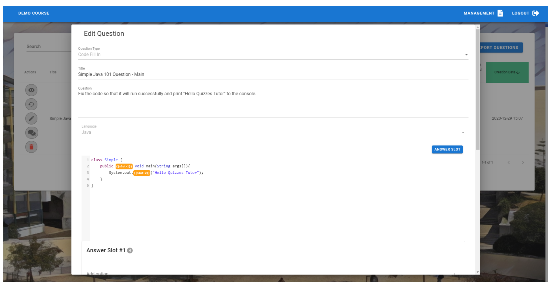
Answer Slot #2

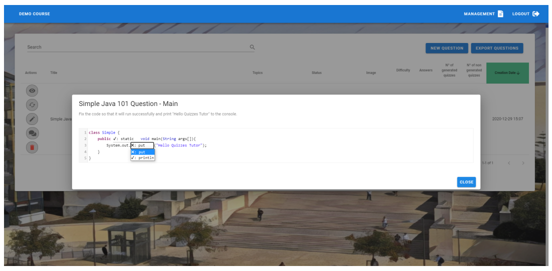
Add option

put

println

(a) Create UI with some highlights.





(b) Edit UI **(c)** View UI

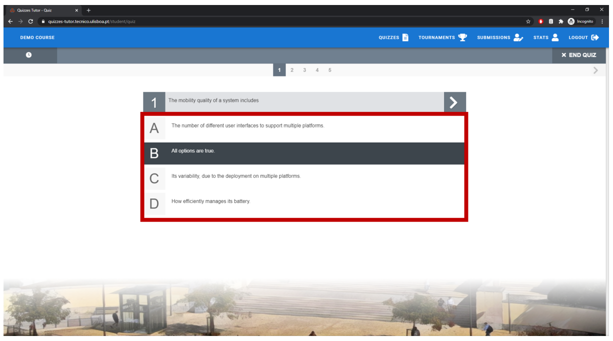
Figure 3.15: Code Fill In Question Management

Secondly, the visualization one, fig. 3.15(c) depicts the new view code question UI. This interface is a read-only interface, where the teacher can see the question and the options like what the student will then see. The code editor is once again using CodeMirror for the highlighting.

All of these UIs for question management are used in multiple places throughout the frontend, being that the management page the most relevant. The components used in the question management are the `CodeFillInCreate.vue` and the `CodeFillInView.vue`. The first for edition, duplication and creation and the second for visualisation of the question.

3.4.4.C Students' interface to answer and validate it

It was necessary to add the question to the students' quiz visualisation UI. To ensure that the UX would be similar to what it was already presented to the students, the code fil-in interface was implemented in the red box (see fig. 3.16), replacing what would be the options for a multiple-choice question. This choice allows for a consistent approach into adding new question types in the future, and only this red-box needs to be implemented for each question. Figure 3.17 presents the UIs implemented for this new question type. This red-box basically represents the answer (`CodeFillInAnswer.vue`) and answer response (`CodeFillInAnswerResponse.vue`) components.



1 The mobility quality of a system includes

A The number of different user interfaces to support multiple platforms.

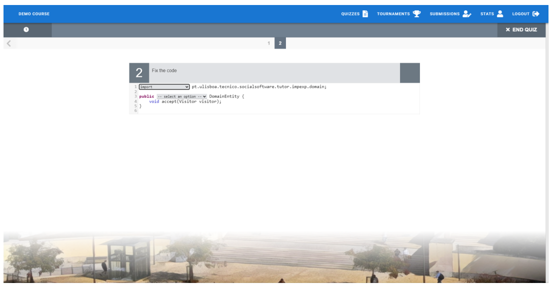
B All options are true.

C Its variability, due to the deployment on multiple platforms.

D How efficiently manages its battery.

Figure 3.16: Multiple choice question, with the `MultipleChoiceAnswer.vue` component identified in red

The final result of this implementation is depicted in fig. 3.17. Note that it draws inspiration from the initial samples done (see Sample implementation of exam code questions sub-section). This UI allows for an intuitive interaction by the learners.

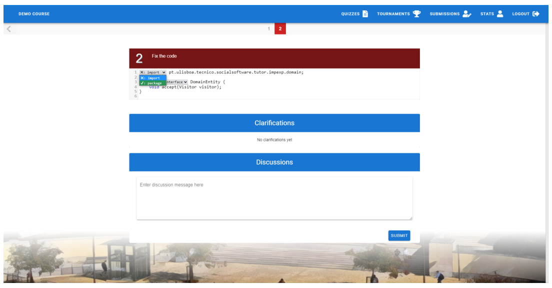


2 Fix the code

```
[import ▾] pt.ulisboa.tecnico.socialsoftware.tutor.impexp.domain;

public [-- select an option -- ▾] DomainEntity {
    void accept(Visitor visitor);
}
```

(a) Code fill in question example



2 Fix the code

```
[X: import ▾] pt.ulisboa.tecnico.socialsoftware.tutor.impexp.domain;
 X: import
 ✓: package      interface ▾] DomainEntity {
    void accept(Visitor visitor);
}
```

No clarifications yet

Enter discussion message here

(b) Code fill in answer example

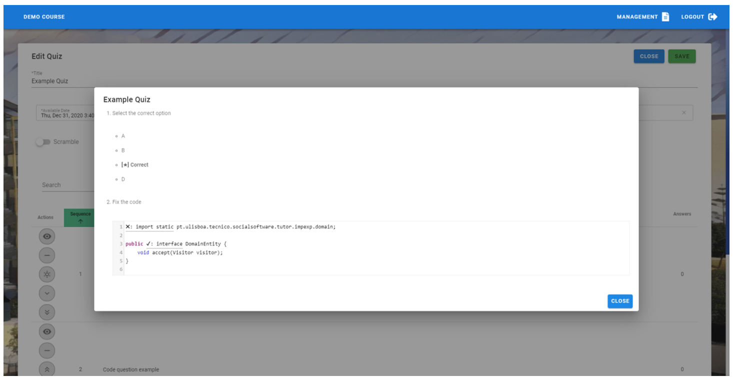
Figure 3.17: Code fill in question type used in quiz

3.4.4.D Teachers' interface to manage quizzes

The interface to manage the quizzes was straightforward up to a certain point. Firstly, quizzes' creation and management deal mostly with the Question domain concept rather than specific questions types.

This means that whenever a new question type is devised, it can be easily added to a quiz without any FE changes. However, some features would not work immediately—namely, the visualisation of the question in a read-only mode and the student's visualisation answer to the question. Both can be solved by ensuring that the question Vue.js specific component is being used. The visualisation component was created when the section 3.4.4.B was developed. The component used here is the `CodeFillInView.vue`.

Figure 3.18 portraits the multiple interfaces used to manage quiz questions and how it is affected by this code question. Figure 3.18(a) shows how all questions can be viewed before creating the quiz. Whilst, Figures 3.18(b) and 3.18(c) present the UI for quiz answer results. Note that before, only the first one existed since multiple choices are direct, either the student answers with the correct key or not. However, with the addition of code fill-in question, this becomes more complex. On the first screen (fig. 3.18(b)) the student will be able to see a high-level overview of the students answer, and see how many of the fill-in spots they have answered correctly. When pressing the result, a new dialog will appear. This dialog(fig. 3.18(c)) allows the teacher to explore each of the errors shown in more detail.


Example Quiz

1. Select the correct option

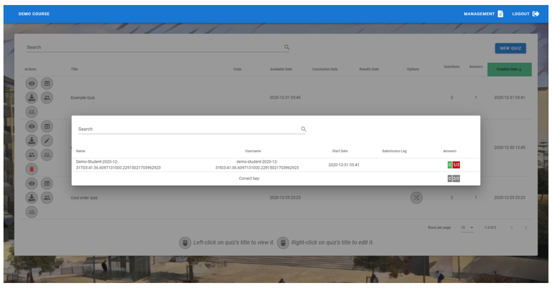
- A
- B
- [★] Correct
- D

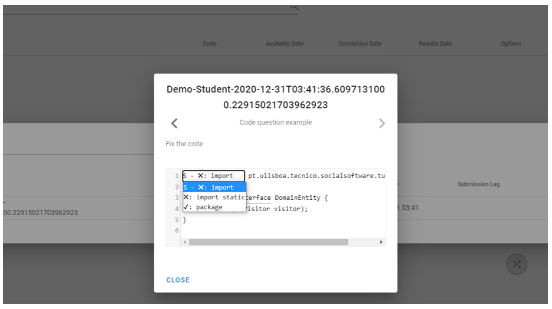
2. Fix the code

```
✗: import static pt.ulisboa.tecnico.socialsoftware.tutor.impexp.domain;

public ✓: interface DomainEntity {
    void accept(Visitor visitor);
}
```

(a) Quiz creation and question visualisation


Name	Username	Start Date	Submission Lag	Answers
Demo-Student-2020-12-31T03:41:36.6097131000.22915021703962923	demo-student-2020-12-31t03:41:36.6097131000.22915021703962923	2020-12-31 03:41		C 1/2
	Correct key:			C 2/2


Demo-Student-2020-12-31T03:41:36.609713100 0.22915021703962923

Code question example

Fix the code

```
S - ✗: import  pt.ulisboa.tecnico.socialsoftware.tu
S - ✗: import
✗: import static terface DomainEntity {
✓: package        isitor visitor);
}
```

(b) Quiz answer results

(c) Quiz answer details

Figure 3.18: Code fill in question type used in quiz

3.4.4.E Frontend changes required to add a new question

As noted from the previous sub-sections, the frontend code became very modular and easy to extend up to a certain degree. The complexity of extension lies with the complexity of the question UI. Figure 3.19 presents a schema of the steps and code required to update the FE code, demonstrating how straightforward it is to do it. As we can see, and reinforced by the previous sections, little additions are to be done.

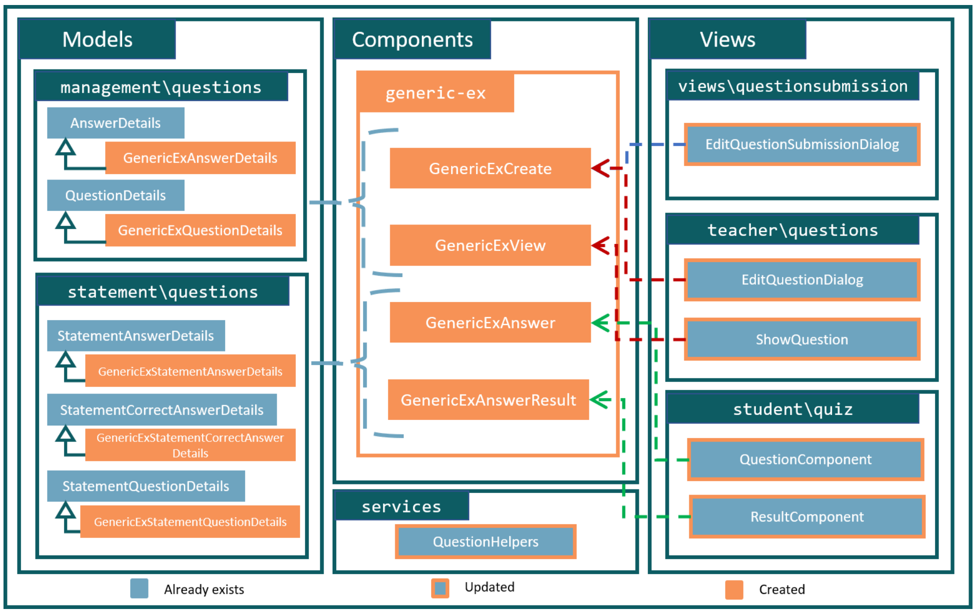


Figure 3.19: Frontend changes required to introduce the new generic question (named: GenericEx)

Once again, the schema just displays the required changes. Any tricky question will probably require more classes or components to allow code re-usage.

4 Evaluation

Contents

With the requirements in mind, it is crucial to stop and critically analyse the work done. The following chapter focuses on just that.

It starts by doing a performance analysis, comparing the original solution with multiple choice questions only, with the new one that allows multiple question types.

Then exposes an examination on the ease of adding a new question type, to ensure that it meets the requirements of creating a new question type quickly and particularly a new code question type, per example the Parsons Problem.

4.1 Performance Comparisons with original solution

With the transformations done to the domain, it is expected that there might exist some impact on performance. However, if strongly affected, this point can harm the current solution, making it nearly unusable or at best very unstable. To ensure that the performance changes are sustainable multiple tests were run against the two multiple implementation versions, before and after the significant domain changes. On one end, we have the original implementation where only multiple-choice existed, and the domain was more straightforward, having `Questions` and `Answers` as the simple domain. On the other end, we have the new domain implementation which contains the existent multiple-choice questions but now as `QuestionsDetails` and `AnswersDetails` of `Questions` and `Answers` respectively.

To perform these tests straightforwardly and systematically, the tests were set up considering the following conditions throughout each iteration of each run:

- All tests ran on the same machine.
- The tests were all created from existing/modified scripts used previously to analyse the code performance.
- The tests mentioned above are executed using JMeter[1] (version 5.2.1).
- Each test starts from the same starting point — an empty database, with the tables created by the Spring migrations.
- Besides that all demo courses required in the tests are created.

As for the actual tests, there are two main tests. The first focus on the creation of questions by the teachers. The second on a regular quiz interaction, starting from creation until the actual answering of the said quiz. Both of these tests are stress tests that allow the comparison of the implementation in some extreme scenarios. Like noted the tests run in the same machine, some small fluctuations

[1] https://jmeter.apache.org/

of values might exist due to other applications running in the machine. The next sections present the results of each test as well as an analysis of the obtained data.

4.1.1 Question Creation

The first batch of tests focuses on creating the questions, to ensure a comparative value it was done using the same question type, and therefore focusing mostly on the domain changes and their possible repercussions.

The creation of questions test consists of login in as a teacher or multiple teachers of a given course and creating a given number of questions. This test exposes if there are any particular issues in the question creation. This is particularly relevant in our case since there were breaking changes that could have broken something or made it substantially slower.

4.1.1.A Fifty teachers to fifty questions (with teardown)

The first test consists of fifty teachers login in their demo account, and each one of them creating five questions. This test was run trice, and after each run, the database questions data was cleaned up as part of the test. By doing this, we are only focused on the most superficial interaction possible, not yet considering what the increased amounts of data might originate.

On table 4.1 and table 4.2, we can see the results of the test to the original implementation and the new one respectively. As we can see, the data is relatively similar, showing a slight improvement on the new implementation.

Table 4.1: Results: 50 Teachers Create 5 questions each (3 Runs with tear down) - Original implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as Demo Teacher	150	266	27	765	181,99	4,16043
HTTP Request to create question	750	408	9	1484	231,03	19,71194
All Requests	900	384	9	1484	229,75	23,62701

Table 4.2: Results: 50 Teachers Create 5 questions each (3 Runs with tear down) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as Demo Teacher	150	217	26	478	128,01	6,19885
HTTP Request to create question	750	375	11	767	210,14	28,29975
All Requests	900	349	11	767	207,36	33,92002

4.1.1.B Fifty teachers to fifty questions (without teardown)

The second question creation test consists of fifty teachers login in their demo account, and each one of them creating five questions, like the first one. This test was also run trice; however, no changes in the database and questions were created, followed by the other. By doing this, we are considering how increased amounts of data might affect the final results.

On table 4.3 and table 4.4, we can see the results of the test to the original implementation and the new one respectively. On this test, we can clearly see a decay in performance as the number of questions increases in the database. These results are not ideal, not for the old implementation nor the new, that being said, this condition is currently improbable where we have so many teachers creating so many questions at the same time. This issue is something that might require some further attention, but as of now, we can consider these results relatively acceptable, especially since the throughput is not that affected. Nevertheless, the now 300-millisecond gap should be shortened.

Table 4.3: Results: 50 Teachers Create 5 questions each (3 Runs without tear down) - Original implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as Demo Teacher	150	1069	30	4072	832,81	6,07780
HTTP Request to create question	750	1463	14	7865	1287,52	19,84652
All Requests	900	1397	14	7865	1232,30	23,79693

Table 4.4: Results: 50 Teachers Create 5 questions each (3 Runs without tear down) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as Demo Teacher	150	1101	25	6087	992,90	5,46110
HTTP Request to create question	750	1785	10	9706	1448,07	17,99986
All Requests	900	1671	10	9706	1405,93	21,58222

4.1.2 Quiz interaction

The second and final batch of tests focuses on quiz capabilities of the system, more concretely the creation, answering and processing. To ensure a comparative value, the tests were done using the same question type, and therefore focusing mostly on the domain changes and their possible repercussions.

The creation of quiz interaction test consists of login as a teacher and creating a given number of questions for a given quiz and the respective quiz. Then a supplied number of students will log in and answer said quiz. After they all submit the grade calculation will be performed. This test exposes if there are any particular issues with the main functionality of Quizzes Tutor, the quizzes. This is particularly relevant in our case since there were breaking changes, both in the questions and the answers, that could have broken something or made it considerably hindered.

For each test, we measured: the quiz creations, with specific questions, the students' answer and quiz submission, and finally, the quiz finalisation where the answers are processed.

4.1.2.A 100 students answer 20 questions

The first test consists of creating a quiz with twenty brand-new questions and making a hundred students answer said quiz simultaneously, finalising with the answer processing.

The first step of the test consists of creating the quiz by a teacher of the demo course. As we can see in table 4.5 and table 4.6, and supported by the previous results, the creation of questions and quizzes are similar in both implementations, which supports the preservation of the solution performance for this use case. Note that, even being the same operations, these results are significantly more performant than the tests done in sec. 3, this can be easily explained with the amounts of requests being performed. With an increase in requests, it is expected that there are impacts.

Table 4.5: Results: Quiz Creation (20 Questions) - Original implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as teacher	1	37	37	37	0,00	27,02703
Create question	20	36	25	147	26,32	26,88172
Get list of questions	1	101	101	101	0,00	9,90099
Create quiz	1	110	110	110	0,00	9,09091
All Requests	23	42	25	147	31,28	23,09237

Table 4.6: Results: Quiz Creation (20 Questions) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as teacher	1	36	36	36	0,00	27,77778
Create question	20	37	25	174	31,91	26,66667
Get list of questions	1	103	103	103	0,00	9,70874
Create quiz	1	106	106	106	0,00	9,43396
All Requests	23	43	25	174	35,25	23,06921

The second step focusses on the students part of the interaction. A hundred students are theoretically answering their answers as fast as possible on this test, causing a high concurrency. Looking at the data from table 4.7 and table 4.8 is possible to note that once again, the two solutions are similar, at least for this test load. On average the new implementation was slightly faster overall achieving as well a slightly larger throughput. These results just go to show the consistency of the new solution when comparing with the previous.

Table 4.7: Results: Students answering the quiz simulation (20 Questions + 100 Students) - Original implementation

Label	# Samples	Average (ms)	Median(ms)	Min (ms)	Max(ms)	Throughput (req/sec)
Login as student	100	1059	1016	98	3994	21,17747
Get quizzes available	100	1513	1341	39	4375	16,86910
Start quiz	100	1695	1604	561	4810	14,94322
Submit answer	2000	225	173	9	2327	243,42746
Conclude quiz	100	185	154	10	818	53,76344
All Requests	2400	373	192	9	4810	241,37584

Table 4.8: Results: Students answering the quiz simulation (20 Questions + 100 Students) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Median(ms)	Min (ms)	Max(ms)	Throughput (req/sec)
Login as student	100	1082	1001	133	4877	18,12251
Get quizzes available	100	1175	1080	95	4172	17,61184
Start quiz	100	1707	1595	609	3577	16,11863
Submit answer	2000	230	173	9	3243	265,25199
Conclude quiz	100	167	131	17	529	51,92108
All Requests	2400	363	190	9	4877	245,29845

The final step considers all the answers provided, currently in a logged state, and saves them properly in the database. This process is a heavy operation that involves the creation of multiple entities. The results present in the table 4.9 and table 4.10, display an unfavorable scenario for the new implementation. Here the performance is hindered in comparison with the original solution. This result was expected since the number of entities required to maintain this new solution is larger than the old hence more data needs to be created. This operation is mostly a management operation usually done at night; hence, the existing platform's impact is diminished. That being said, this could become a problem in the future and can become more evident with the increase of users, should be developed a solution that tries to optimise this situation.

Table 4.9: Results: Quiz finalisation (20 Questions each 100 Answers) - Original implementation

Label	# Samples	Average (ms)	Throughput (req/sec)
Login as teacher	1	73	13,69863
Write quiz answers	1	3891	0,25700
All Requests	2	1982	0,50441

Table 4.10: Results: Quiz finalisation (20 Questions each 100 Answers) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Throughput (req/sec)
Login as teacher	1	66	15,15152
Write quiz answers	1	5587	0,17899
All Requests	2	2826	0,35373

4.1.2.B 300 students answer 20 questions

The second test consists of creating a quiz with twenty brand-new questions and using three hundred students answering said quiz simultaneously, finalising with the answer processing. The goal of this scenario is trying to take the platform to the limits of one of the largest courses in our university.

The first step of the test consists of creating the quiz by a teacher of the demo course. As we can see in table 4.11 and table 4.12, and supported by the previous results, the creation of questions and quizzes are similar in both implementations, which supports the preservation of the solution performance for this use case. Note that this test set is basically the same as the one in the last batch, presenting therefore also similar results.

Table 4.11: Results: Quiz Creation (20 Questions) - Original implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as teacher	1	85	85	85	0,00	11,76471
Create question	20	25	18	38	5,35	38,61004
Get list of questions	1	61	61	61	0,00	16,39344
Create quiz	1	94	94	94	0,00	10,63830
All Requests	23	32	18	94	19,55	30,26316

Table 4.12: Results: Quiz Creation (20 Questions) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as teacher	1	88	88	88	0,00	11,36364
Create question	20	31	21	45	5,27	31,25000
Get list of questions	1	121	121	121	0,00	8,26446
Create quiz	1	99	99	99	0,00	10,10101
All Requests	23	41	21	121	24,85	24,18507

Once again, the second step focusses on the students part of the interaction. In this particular test, three hundred students are theoretically answering their answers as fast as possible on this test, causing a high concurrency. Looking at the data from table 4.13 and table 4.14 is possible to note that once again, the two solutions are similar, at least for this test load. This result even with a larger amount of students than the previous one produced similar results. On average the new implementation was slightly faster overall achieving as well a slightly larger throughput. These results are good for the new solution, especially because it shows this consistency throughout the tests when it comes to students interaction and UX, which is the most important part. Nevertheless, these results are not that fantastic that should be elevated to that level.

Table 4.13: Results: Students answering the quiz simulation (20 Questions + 300 Students) - Original implementation

Label	# Samples	Average (ms)	Median(ms)	Min (ms)	Max(ms)	Throughput (req/sec)
Login as student	300	5392	4601	465	24579	11,62565
Get quizzes available	300	7524	7443	1505	20480	10,71965
Start quiz	300	6194	5455	1794	19470	11,16778
Submit answer	6000	958	560	10	14471	205,33178
Conclude quiz	300	395	349	16	1406	63,93862
All Requests	7200	1611	615	10	24579	175,76409

The final step considers all the answers provided, currently in a logged state, and saves them properly in the database. This process is a heavy operation that involves the creation of multiple entities. Unfortunately, the results present in the table 4.15 and table 4.16, display an unfavorable scenario, once again, for the new implementation. With the increase in students and therefore, in answers, the time

Table 4.14: Results: Students answering the quiz simulation (20 Questions + 300 Students) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Median(ms)	Min (ms)	Max(ms)	Throughput (req/sec)
Login as student	300	5354	4503	375	21627	13,40183
Get quizzes available	300	7161	6818	207	24420	11,17985
Start quiz	300	6199	5477	1417	20701	10,07692
Submit answer	6000	935	568	10	14156	209,04467
Conclude quiz	300	409	324	15	1917	51,01173
All Requests	7200	1576	621	10	24420	180,08554

that takes to process them is also increasing. Worst than that is the new solution's performance which continues significantly slower compared to the original solution. However, as stated before this results are expected for the same reasons explained before.

Table 4.15: Results: Quiz finalisation (20 Questions each 300 Answers) - Original implementation

Label	# Samples	Average (ms)	Throughput (req/sec)
Login as teacher	1	227	4,40529
Write quiz answers	1	10518	0,09508
All Requests	2	5372	0,18613

Table 4.16: Results: Quiz finalisation (20 Questions each 300 Answers) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Throughput (req/sec)
Login as teacher	1	216	4,62963
Write quiz answers	1	14709	0,06799
All Requests	2	7462	0,13400

4.1.2.C 300 students answer 40 questions

The third and final test consists of creating a quiz with forthy brand-new questions and using three hundred students answering said quiz simultaneously, finalising with the answer processing. On this last scenario, besides increasing the number of students, the number of questions was also increased, to see which impact this increase might have.

Once again, the first step of the test, which consists of creating the quiz by a teacher of the demo course, presented a favourable result. As we can see in table 4.17 and table 4.18, and continuously supported by the previous results, the creation of questions and quizzes are similar in both implementations. This combinations of tests are pretty similar to what could happen in reality, so it is very positive that the results maintain theses stable throughout the tests.

As seen previously, the second step focusses on the students part of the interaction. Looking at the data from table 4.19 and table 4.20 is possible to note that again, the two solutions are similar. This result even with a larger amount of students than the previous one produced similar results. However,

Table 4.17: Results: Quiz Creation (40 Questions) - Original implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as teacher	1	199	199	199	0,00	5,02513
Create question	40	35	27	49	5,58	27,83577
Get list of questions	1	110	110	110	0,00	9,09091
Create quiz	1	86	86	86	0,00	11,62791
All Requests	43	42	27	199	28,11	23,43324

Table 4.18: Results: Quiz Creation (40 Questions) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev. (ms)	Throughput (req/sec)
Login as teacher	1	197	197	197	0,00	5,07614
Create question	40	37	27	53	6,25	26,49007
Get list of questions	1	128	128	128	0,00	7,81250
Create quiz	1	77	77	77	0,00	12,98701
All Requests	43	44	27	197	28,41	22,46604

with the increase of questions this time, the original solution behaved better, achieving better results. Combined with the previous one, these results present a good scenario for the new solution showing how similar this new solution can be performance-wise.

Table 4.19: Results: Students answering the quiz simulation (40 Questions + 300 Students) - Original implementation

Label	# Samples	Average (ms)	Median(ms)	Min (ms)	Max(ms)	Throughput (req/sec)
Login as student	300	12610	11394	1773	43693	6,62530
Get quizzes available	300	14245	12862	2674	46272	5,62746
Start quiz	300	10780	9426	3243	34391	6,17996
Submit answer	12000	950	523	10	23907	209,87460
Conclude quiz	300	370	295	12	1336	46,27487
All Requests	13200	1728	548	10	46272	167,49994

Table 4.20: Results: Students answering the quiz simulation (40 Questions + 300 Students) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Median(ms)	Min (ms)	Max(ms)	Throughput (req/sec)
Login as student	300	12757	11294	2256	45572	6,35055
Get quizzes available	300	15180	13762	2677	52026	5,02378
Start quiz	300	13444	12192	3383	40227	5,51005
Submit answer	12000	1024	548	10	31742	197,35544
Conclude quiz	300	357	269	11	1416	54,91488
All Requests	13200	1879	569	10	52026	155,10981

Finally, to wrap up the tests, and this batch precisely, the third step which considers all the answers provided, currently in a logged state, and saves them properly in the database. Once again, we can see some detrimental performance. These results were as explained before expected, but once again, it emphasises how important it is to take a second look at this data and improve upon these results.

Table 4.21: Results: Quiz finalisation (40 Questions each 300 Answers) - Original implementation

Label	# Samples	Average (ms)	Throughput (req/sec)
Login as teacher	1	413	2,42131
Write quiz answers	1	22679	0,04409
All Requests	2	11546	0,08661

Table 4.22: Results: Quiz finalisation (40 Questions each 300 Answers) - Multiple Question Types implementation

Label	# Samples	Average (ms)	Throughput (req/sec)
Login as teacher	1	377	2,65252
Write quiz answers	1	31782	0,03146
All Requests	2	16079	0,06219

4.2 Ease of adding a new code question

One of the requirements previously stated (see section 3.2) was to ensure a flexible architecture. On other words, with the addition of this new code question and domain reconfiguration, the code should result in a low development cost when adding new types of problems.

Considering the steps to add a new question, as proposed in the solution, both for the frontend (section 3.4.4.E) and the backend (section 3.4.3.C), it is relatively straightforward to accomplish it. To put the requirements to the test, and to ensure that the new solution provides a low development cost when adding new types of problems.

This test was set up considering the following conditions:

- It will be implemented a new code question. After considering the possibilities, it was decided that an ordering code question would be implemented, like the Parsons Problem [33].
- The author did the developments.
- Each development period was timed to grasp the time taken in each part of the development.
- It will be analysed Lines Of Code (LOC), altered classes, as well as newly created classes.

The following sub-sections present the transformations done. Starting with the BE side and then focusing on the FE. Finally, it ends with a summary analysing the overall effort of adding the new code question.

4.2.0.A Backend changes for new question evaluation

Following the recommendations from section 3.4.3.C it was elementary to start implementing and adding this new question.

Firstly, as for any development, it was necessary to understand the problem. The new question to be added is a Parsons Problem. This means that it is an ordering question. Denoting each section to be ordered as `Slot`, it starts to become clear that the question will have a list of slots. It is also important to note that not all slots are part of the solution and are only distractors. To simplify the domain, we consider that a `Slot` has `Order` and if `null` it means that is not part of the answer. With this in mind, the domain started to form, resulting in fig. 4.1.

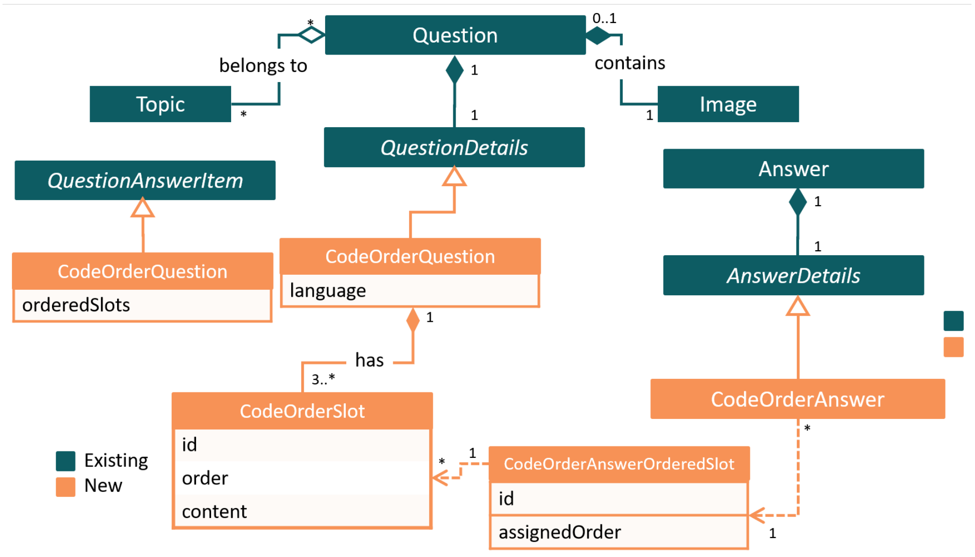


Figure 4.1: Question, QuestionAnswer and QuestionAnswerItem domain model, after the introduction code order question

Table 4.23 below lists all changes that occurred on the backend, indicating addition or updates.

All these changes were really mechanical, as soon as the domain is clear the remainder changes become trivial. The development time for all the changes above with tests included was 2 hours, 1h35 for the initial development, and the remainder of the time spent with fixes while developing the FE.

4.2.0.B Frontend changes for new question evaluation

With the BE, developed and tested, it was possible to start developing the FE. Taking similar steps as the solution (see section 3.4.4), the first step was to map the created DTOs. It was also created a skeleton for the future components, and the question management was updated to be ready to deserialise the new question correctly.

Starting with the questions, then the students part and finally the quizzes part, the solution started to come together. In fig. 4.2 we can see the interface for creating new order questions, it is displayed together with the other questions, making it easy to use. A teacher will start with three slots, and he can create more, or select which slots should be used for the actual question. Figure 4.3 portraits the question visualisation. Even though it might seem that the question is saved ordered and the same order is always passed around that is not true. When creating a new question slots ids are created in random

Table 4.23: Backend File changes with (C)reated and (M)odified files

Package	Class	LOC	Op
	Answer		
answer.domain	CodeOrderAnswer	+98;-0	C
answer.domain	CodeOrderAnswerOrderedSlot	+62;-0	C
answer.dto	AnswerDetailsDto	+3;-3	M
answer.dto	CodeOrderAnswerDto	+29;-0	C
answer.dto	CodeOrderAnswerOrderedSlotDto	+46;-0	C
answer.dto	CodeOrderCorrectAnswerDto	+27;-0	C
answer.dto	CorrectAnswerDetailsDto	+3;-3	M
	Question		
question	Updator	+9;-2	M
question.domain	CodeOrderQuestion	+145;-0	C
question.domain	CodeOrderSlot	+85;-0	C
question.domain	Question	+1;-0	M
question.dto	CodeOrderQuestionDto	+65;-0	C
question.dto	CodeOrderSlotDto	+44;-0	C
question.dto	QuestionDetailsDto	+3;-3	C
	Statement		
statement.domain	CodeOrderAnswerItem	+38;-0	C
statement.domain	CodeOrderSlotAnswerItem	+32;-0	C
statement.dto	CodeOrderSlotStatementAnswerDetailsDto	+34;-0	C
statement.dto	CodeOrderSlotStatementQuestionDetailsDto	+31;-0	C
statement.dto	CodeOrderStatementAnswerDetailsDto	+69;-0	C
statement.dto	CodeOrderStatementQuestionDetailsDto	+42;-0	C
statement.dto	StatementAnswerDetailsDto	+2;-2	M
statement.dto	StatementQuestionDetailsDto	+2;-2	M
	Other		
exceptions	ErrorMessage	+3;-0	M
impexp.domain	Visitor	+4;-0	M
impexp.domain	CSVQuizExportVisitor	+14;-2	M
impexp.domain	LatexVisitor	+31;-4	M
impexp.domain	QuestionsXmlImport	+28;-1	M
impexp.domain	XMLQuestionExportVisitor	+27;-0	M
Summary	28 files changed, 977 insertions(+), 22 deletions(-) 16 files (C)reated, 12 files (M)odified		

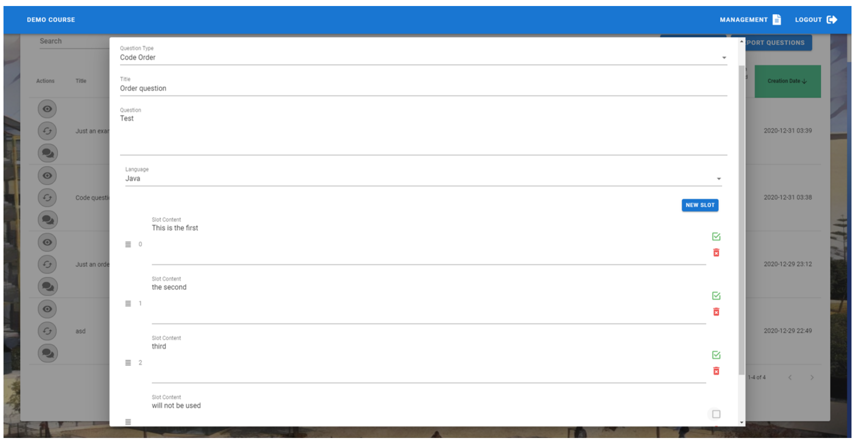
order. To simplify our life, the management views always receive the slots ordered by order of the used ones and then followed by the not used in the question.

Finally, fig. 4.4 presents the UIs as seen by the students. Here they can drag from the available options to the answer slot to create the correct answer. In order to achieve, these draggable functionalities the vuedragable[2] was used.

Similarly to what was done on the BE, the table 4.24 summarises all the changes made on the FE side.

This development took more time on the frontend part than the backend. The approximate time taken was 7 hours, which is considerably more than the backend part, which is easily explained by the fact

[2]https://www.npmjs.com/package/vuedraggable



Question Type
Code Order

Title
Order question

Question
Test

Language
Java

0 Slot Content
This is the first

1 Slot Content
the second

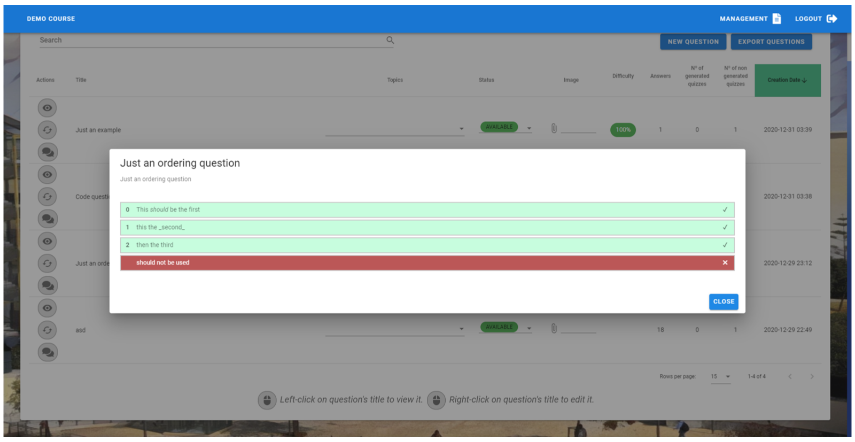
2 Slot Content
third

Slot Content
will not be used

Title	Creation Date
Just an exar	2020-12-31 03:39
Code questi	2020-12-31 03:38
Just an orde	2020-12-29 23:12
asd	2020-12-29 22:49

1-4 of 4

Figure 4.2: CodeOrderQuestion create component, used to create a new question



Actions	Title	Topics	Status	Image	Difficulty	Answers	Nº of generated quizzes	Nº of non generated quizzes	Creation Date
	Just an example		AVAILABLE		100%	1	0	1	2020-12-31 03:39
	Code questi								2020-12-31 03:38
	Just an orde								2020-12-29 23:12
	asd		AVAILABLE			18	0	1	2020-12-29 22:49

Just an ordering question

Just an ordering question

0 This *should* be the first ✓

1 this the _second_ ✓

2 then the third ✓

should not be used ✕

Rows per page: 15 1-4 of 4

Left-click on question's title to view it. *Right-click on question's title to edit it.*

Figure 4.3: CodeOrderQuestion view component, used in the question management page

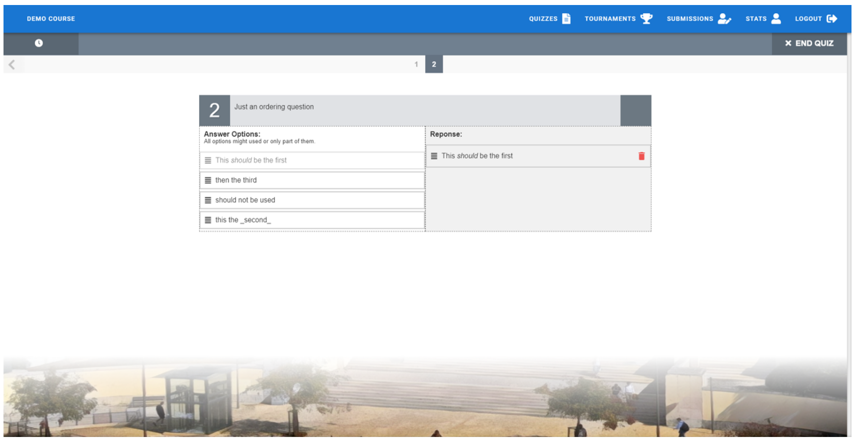
that this development requires more creative sides.

4.2.0.C Analysis of effort

The backend changes are more, in terms of both files changed and created, compared to FE. However, the changes are not complex at all and are somewhat mechanical and repetitive. Also, the time taken for all of these changes, which can be considered low, shows the simpleness. It is not ideal to have to change these many files, mainly because it is easy to forget some of them. Still, they are easily discovered in tests or during the frontend implementation.

On the contrary, frontend changes were relatively low; it is noticeable that the changes on the modified files are trivial. In terms of files created the changes were much more complicated. This will probably happen in all implementations of new question types, where the UI and UX are among the most considerable challenges.

Looking at the data, both files changed and time taken, this solution is up to the challenge of adding



2 Just an ordering question

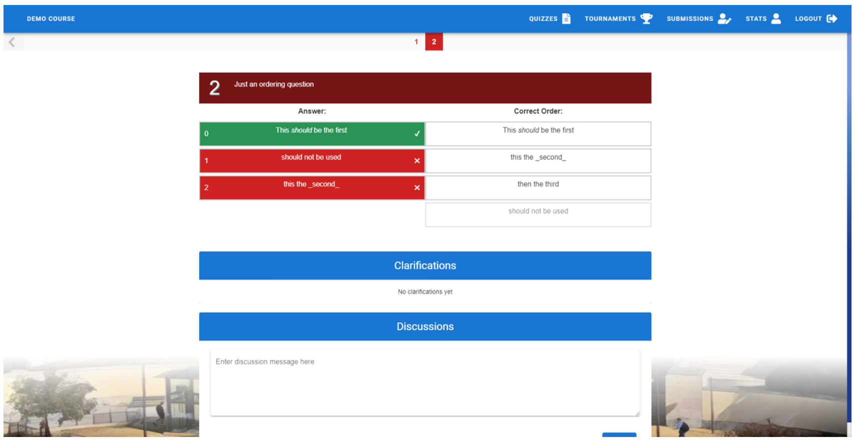
Answer Options:
All options might used or only part of them.

- This *should* be the first
- then the third
- should not be used
- this the _second_

Reponse:

- This *should* be the first

(a) Code order question example



2 Just an ordering question

Answer:	Correct Order:
0 This *should* be the first ✓	This *should* be the first
1 should not be used ✗	this the _second_
2 this the _second_ ✗	then the third
	should not be used

Clarifications

No clarifications yet

Discussions

Enter discussion message here

(b) Code order answer example

Figure 4.4: Code order question type used in quiz, as seen by the students

new questions with ease and low development effort. The only problems that might arise are when more complex questions are selected to be inserted. Similarly to what happened with this question, there might have more complex parts; this case was the front end.

Table 4.24: Frontend File changes with (C)reated and (M)odified files

Path	Class	LOC	Op
	Models		
models/management/questions	CodeOrderAnswerDetails.ts	+41;-0	C
models/management/questions	CodeOrderAnswerOrderedSlot.ts	+13;-0	C
models/management/questions	CodeOrderQuestionDetails.ts	+26;-0	C
models/management/questions	CodeOrderSlot.ts	+17;-0	C
models/statement/questions	CodeOrderSlotStatementAnswerDetails.ts	+11;-0	C
models/statement/questions	CodeOrderSlotStatementQuestionDetails.ts	+11;-0	C
models/statement/questions	CodeOrderStatementAnswerDetails.ts	+34;-0	C
models/statement/questions	CodeOrderStatementCorrectAnswerDetails.ts	+14;-0	C
models/statement/questions	CodeOrderStatementQuestionDetails.ts	+21;-0	C
services	QuestionHelpers.ts	+32;-1	M
	Components		
components/code-order	CodeOrderAnswer.vue	+173;-0	C
components/code-order	CodeOrderAnswerResult.vue	+106;-0	C
components/code-order	CodeOrderCreate.vue	+122;-0	C
components/code-order	CodeOrderSlotEditor.vue	+73;-0	C
components/code-order	CodeOrderView.vue	+77;-0	C
	Views		
views/questionsubmission	EditQuestionSubmissionDialog.vue	+3:-1	M
views/student/quiz	QuestionComponent.vue	+3:-1	M
views/student/quiz	ResultComponent.vue	+3:-1	M
views/teacher/questions	EditQuestionDialog.vue	+3:-1	M
views/teacher/questions	ShowQuestion.vue	+3:-1	M
	Other		
configuration	package-lock.json	+28;-0	M
configuration	package.json	+1;-0	M
Summary	22 files changed, 815 insertions(+), 6 deletions(-) 14 files (C)reated, 8 files (M)odified		

5 Conclusion

Contents

Education is in constant evolution. Adapting to the times, the knowledge that needs to be shared, and with new tools that facilitate this knowledge transfer. Throughout this work was described Quizzes Tutor an instrument to help teachers reach students. Also, an extension of the tool, that helps teachers prepare even more engaging questions and prepares the platform to receive new and exciting question types.

This final chapter presents the conclusion of this work. It summarises and reflects on the research performed and highlights the contributions done to Quizzes Tutor. Concludes offering a critical evaluation of the work done in this thesis.

Finally, it displays some system limitations, that can be worked upon to create an even better tool. Also, it underlines the future work to be performed.

5.1 Conclusions

Throughout the years, researchers have been exploring the best tools and methodologies to ensure that, as a teacher, the message passes down successfully. The research performed clearly highlights the advantages of using quizzes, as a learning tool, as they can boost attention levels and knowledge retention. Adding to that, it is becoming clear that the E-Learning concept is nowadays more present than ever. With multiple platforms available, each specialised in a specific niche, having knowledge and the ability to learn on the tip of our hands is enormous. This year particularly, with the pandemic, recuring to these online platforms become even more inevitable. All this information makes Quizzes Tutor very relevant.

Even though education is sometimes challenging, it can be simplified through the usage of the correct tools. More precisely, programming education has been relatively challenging, leading researchers to create effective tools that can help in this effort. Throughout the research explored during this thesis, we can see multiple engaging tools. Platforms that support the gamification of programming exercises, other programs that help teachers execute the code and evaluate it automatically and systematically. Also, simpler mechanisms for teaching programming like the Parsons Problem, which is a programming puzzle in all its essence.

This thesis focusses on learning from related work in this and parallel fields of study, and with that understanding help Quizzes Tutor grow. Using the best practices of design patterns and DDD to continuously evolve the platform. One of the more significant contributions of this work was the reconstruction of the domain (aborded in section 3.3). This transformation allows for the platform to continue its growth with ease. It is a solution built on top of best practices that aims to be flexible and easy to use while maintaining similar performance levels. It is important to note that there where some hindered performance in a particular test. Although it is not enough to discredit the current solution, it is undoubtedly a red-flag that should be evaluated—followed by an improvement.

This thesis's main focus was unquestionably the addition of a new programming code question to the Quizzes Tutor platform. During this work, there was the evaluation of multiple systems, but even more relevant was some analysis already done on a couple of question types that could eventually be added, programming related or not. This analysis covered question goals, question evaluation, describing some of the challenges of implementing it (see sections 2.3.7 and 3.4.1).

The most meaningful contribution of this thesis was the addition of a new programming code question, the Fill-In Question. This code question consists of having a snippet of code with blank spaces that need to be filled. This question arose from the previous evaluation of existing solutions and exam questions. The addition of this question served two purposes. Firstly, the addition of the actual question, which leads to the implementation of programming code support in the frontend. Secondly, and more importantly, it concretises the flexible architecture by fixing any missing details from the domain's initial transformation. This architecture results in a low development cost of adding new types of question, which was one of the most important requisites. A new question implementation was put to the test to prove the flexibility of the new code structure (see section 4.2). The new question was successfully built with ease in roughly eight hours. However, there were quite some code changes, which are explained by two reasons, specific code required to implement the new code, and boilerplate code that can be generated automatically.

In regards to evaluation, there was a deficit that was not filled. One of the shortcomings of this work is not being tested with live users and not gathering information from users in a structured manner. It would have been essential to measure acceptance of the new solution and put to the test with quizzes and exams. From the user tests, two relevant approaches should have been taken into consideration. Firstly, the students would have been essential to see how well they would appreciate these changes and new questions. It would also be very relevant to compare grades compared to regular exams and engagement levels with the material. Secondly, it would be relevant to evaluate the teachers' side. Their evaluation would, of course, focus more on the management part of the questions and quizzes. These feedbacks were partially obtained during the development, but never in a structured way, making it useless for data analysis and evaluation. This lack of analysis is, without a doubt, the biggest shortcoming of this work. Hopefully could be fixed shortly, and make use of those inputs to continue to develop the application.

Quizzes Tutor, grew a lot during this last year. This work showcases part of those developments. Quizzes Tutor is becoming a powerful E-Learning tool, combining the question varieties with engagement mechanisms, like tournaments. This work will undoubtedly encourage even more developments, and help achieve more use cases. Education is in constant evolution, and Quizzes Tutor aims to accompany said evolution as a powerful tool companion to education.

5.2 System Limitations and Future Work

Considering the work developed, the experiences and the shortcomings, there are four principal areas to work in the future. All of the areas achievable independently or complementary. The following listing presents the suggested areas of improvement:

- Continue introducing new question types - especially programming questions. During the development of this work, there was always a focus on laying a path for the future since not all could be achieved. With the analysis done as a starting point and this new infrastructure, it is now possible to start envisioning new questions focusing on more use cases. Particularly, it would be very interesting to introduce questions like the code review or code challenge when it comes to programming questions. However, and as stated before, this will also require a specialised sandbox system to run said tests. This addition would be a great tool, more than evaluating even for learning and serving as a self-assessment tool.

- Do a formal review of the current platform. Quizzes tutor grew a lot during this last year. It would be fascinating to have a formal analysis of how it affects the students and teachers. This analysis would, without a doubt result in exciting action points that can be the focus of future works.

- Automate question creation. As highlighted during the evaluation, there are many classes required and created to introduce a new question. However, this is mostly boilerplate code, that could easily be automatically created. Note that this would not reduce the number of changes, but would reduce the number of changes that are the developer's responsibility, hence being less error-prone. A structured tool that can take a configuration file, such as a YAML, and transform the source code recurring to inheritance to create its files and other methodologies would be great, not only for this scenario.

- Continue extending Quizzes Tutor. Multiple ideas can be introduced into Quizzes Tutor that can help take it to the next level.
  - Answer fraud detection - live and post-answer
  - Lecture materials - that can be complemented with the quizzes or vice-versa
  - Guide learning - analyse the students' difficulties and recommend learning materials and study areas to focus on.

Bibliography

[1] W. Horton, *E-learning by design*. John Wiley & Sons, 2011.

[2] S. Downes, “E-learning 2.0,” *ELearn*, vol. 2005, no. 10, p. 1, 2005.

[3] A. M. Bodzin and W. M. Cates, “Enhancing Preservice Teachers’ Understanding of Web-based Scientific Inquiry,” *Journal of Science Teacher Education*, vol. 14, no. 4, pp. 237–257, 2003.

[4] M. I. Santally and J. Raverdy, “The Master’s Program in Computer-Mediated Computer Communications: A Comparative Study of Two Cohorts of Students,” *Educational Technology Research and Development*, vol. 54, no. 3, pp. 312–326, 2006.

[5] “Quizzes Tutor Website.” [Online]. Available: https://quizzes-tutor.tecnico.ulisboa.pt/

[6] “Quizzes Tutor Source Code.” [Online]. Available: https://github.com/socialsoftware/quizzes-tutor

[7] P. Correia, “Development and evolution of e-assessment platform based on multiple choice questions,” Master’s thesis, Instituto Superior Técnico, University of Lisbon, Instituto Superior Técnico, 9 2020. [Online]. Available: https://fenix.tecnico.ulisboa.pt/cursos/meic-a/dissertacao/283828618790611

[8] T. E. J. Vos, I. S. W. B. Prasetya, G. Fraser, I. Martinez-Ortiz, I. Perez-Colado, R. Prada, J. Rocha, and A. R. Silva, “Impress: Improving engagement in software engineering courses through gamification,” in *Product-Focused Software Process Improvement*, X. Franch, T. Männistö, and S. Martínez-Fernández, Eds. Cham: Springer International Publishing, 2019, pp. 613–619.

[9] T. E. J. Vos, G. Fraser, I. Martinez-Ortiz, R. Prada, A. R. Silva, and I. S. W. B. Prasetya, “Tutorial on a gamification toolset for improving engagement of students in software engineering courses,” in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)*, 2020, pp. 1–3.

[10] J. B. Rocha, L. F. C. Costa, R. Prada, A. R. Silva, D. Gonçalves, and P. Correia, “Quizzes (as a tool for self-regulated learning) in software engineering education,” in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)*, 2020, pp. 1–10.

[11] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, “Review of recent systems for automatic assessment of programming assignments,” *Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli Calling'10*, pp. 86–93, 2010.

[12] E. Evans, *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[13] H. L. Roediger and J. D. Karpicke, “The Power of Testing Memory: Basic Research and Implications for Educational Practice,” *Perspectives on Psychological Science*, vol. 1, no. 3, pp. 181–210, 2006.

[14] A. C. Butler and H. L. Roediger, “Feedback enhances the positive effects and reduces the negative effects of multiple-choice testing,” *Memory and Cognition*, vol. 36, no. 3, pp. 604–616, mar 2008. [Online]. Available: https://link.springer.com/article/10.3758/MC.36.3.604

[15] H. L. Roediger and J. D. Karpicke, “Test-enhanced learning: Taking memory tests improves long-term retention,” *Psychological Science*, vol. 17, no. 3, pp. 249–255, 2006.

[16] M. A. McDaniel, P. K. Agarwal, B. J. Huelser, K. B. McDermott, and H. L. Roediger, “Test-Enhanced Learning in a Middle School Science Classroom: The Effects of Quiz Frequency and Placement,” *Journal of Educational Psychology*, vol. 103, no. 2, pp. 399–414, 2011.

[17] C. Cheong, F. Cheong, and J. Filippou, “Quick quiz: A gamified approach for enhancing learning,” *Proceedings - Pacific Asia Conference on Information Systems, PACIS 2013*, 2013.

[18] G. S. Mason, T. R. Shuman, and K. E. Cook, “Comparing the effectiveness of an inverted classroom to a traditional classroom in an upper-division engineering course,” *IEEE Transactions on Education*, vol. 56, no. 4, pp. 430–435, 2013.

[19] C. Watson and F. W. Li, “Failure rates in introductory programming revisited,” *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*, pp. 39–44, 2014.

[20] J. Bennedsen and M. E. Caspersen, “Failure rates in introductory programming - 12 years later,” *ACM Inroads*, vol. 10, no. 2, pp. 30–35, 2019.

[21] E. Lahtinen, K. Ala-Mutka, and H. M. Järvinen, “A study of the difficulties of novice programmers,” *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 14–18, 2005.

[22] A. Pears, S. Seidman, L. Mannila, L. Malmi, E. Adams, B. Jens, M. Devlin, and J. Paterson, “A Survey of Literature on the Teaching of Introductory Programming,” vol. 1846, pp. 161–180, 2018.

[23] A. Vihavainen, J. Airaksinen, and C. Watson, “A systematic review of approaches for teaching introductory programming and their influence on success,” *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research*, pp. 19–26, 2014.

[24] J. M. Sáez-López, M. Román-González, and E. Vázquez-Cano, “Visual programming languages integrated across the curriculum in elementary school: A two year case study using ”scratch” in five schools,” *Computers and Education*, vol. 97, pp. 129–141, jun 2016.

[25] D. G. Kay, T. Scott, P. Isaacson, and K. A. Reek, “Automated Grading Assistance for Student Programs,” Tech. Rep. 1, 1994.

[26] K. M. Ala-Mutka, “A Survey of Automated Assessment Approaches for Programming Assignments,” *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005. [Online]. Available: https://www.tandfonline.com/action/journalInformation?journalCode=ncse20

[27] R. Lobb and J. Harlow, “Coderunner: A tool for assessing computer programming skills,” Tech. Rep. 1, 2016.

[28] D. Pritchard and T. Vasiga, “CS Circles: An in-browser python course for beginners,” *SIGCSE 2013 - Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, no. December, pp. 591–596, 2013.

[29] “CodeMirror.” [Online]. Available: https://codemirror.net/

[30] J. P. Leal and F. Silva, “Mooshak: A Web-based multi-site programming contest system,” *Software - Practice and Experience*, vol. 33, no. 6, pp. 567–581, 2003.

[31] P. Brusilovsky and S. Sosnovsky, “Individualized Exercises for Self-Assessment of Programming Knowledge: An Evaluation of QuizPACK,” *ACM Journal on Educational Resources in Computing*, vol. 5, no. 3, p. 6, 2005.

[32] I.-H. Hsiao, P. Brusilovsky, and S. Sosnovsky, “Web-based Parameterized Questions for Object-Oriented Programming,” *World Conference on E-Learning, E-Learn 2008*, no. June 2014, pp. 3728–3735, 2008.

[33] D. Parsons and P. Haden, “Parson’s programming puzzles: A fun and effective learning tool for first programming courses,” Tech. Rep., 2006. [Online]. Available: https://www.researchgate.net/publication/262160581

[34] J. Chauhan and A. Goel, “An analysis of quiz in MOOC,” *2016 9th International Conference on Contemporary Computing, IC3 2016*, 2017.

[35] N. Reid and C. McLoughlin, “Designing Online Quiz Questions To Assess a Range of Cognitive Skills,” Tech. Rep., 2002.

[36] M. Chandratilake, M. Davis, and G. Ponnamperuma, “Assessment of medical knowledge: The pros and cons of using true/false multiple choice questions,” Tech. Rep. 4, 2011.

[37] E. L. Bjork, J. L. Little, and B. C. Storm, “Multiple-choice testing as a desirable difficulty in the classroom,” *Journal of Applied Research in Memory and Cognition*, vol. 3, no. 3, pp. 165–170, jul 2014.

[38] EdX, “Building and Running an edX Course,” 2014. [Online]. Available: https://edx.readthedocs.io/projects/edx-partner-course-staff/en/latest/index.html

[39] Oracle, “Java11 Programmer Study Guide.” [Online]. Available: https://www.oracle.com/a/ocom/img/dc/ww-java11-programmer-study-guide.pdf

[40] D. H. Johnson, “Teaching a ’mooc:’ Experiences from the front line,” *2013 IEEE Digital Signal Processing and Signal Processing Education Meeting, DSP/SPE 2013 - Proceedings*, pp. 268–272, 2013.

[41] M. Agarwal and P. Mannem, “Automatic Gap-fill Question Generation from Text Books,” Tech. Rep. June, 2011.

[42] E. Sumita, F. Sugaya, and S. Yamamoto, “Measuring non-native speakers’ proficiency of English by using a test with automatically-generated fill-in-the-blank questions,” Tech. Rep., 2005. [Online]. Available: http://www.ets.org/toefl/

[43] J. Hill and R. Simha, “Automatic Generation of Context-Based Fill-in-the-Blank Exercises Using Co-occurrence Likelihoods and Google n-grams,” Tech. Rep., 2016.

[44] N. Funabiki, Y. Korenaga, Y. Matsushima, T. Nakanishi, and K. Watanabe, “An online fill-in-the-blank problem function for learning reserved words in Java programming education,” in *Proceedings - 26th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2012*, 2012, pp. 375–380.

[45] N. Funabiki, Y. Korenaga, T. Nakanishi, and K. Watanabe, “An extension of fill-in-the-blank problem function in Java programming learning assistant system,” in *2013 IEEE Region 10 Humanitarian Technology Conference, R10-HTC 2013*, 2013, pp. 85–90.

[46] P. Denny, A. Luxton-Reilly, and B. Simon, *Evaluating a new exam question: Parsons problems*, 2008.

[47] B. J. Ericson, L. E. Margulieux, and J. Rick, “Solving parsons problems versus fixing and writing code,” *ACM International Conference Proceeding Series*, pp. 20–29, 2017. [Online]. Available: https://doi.org/10.1145/3141880.3141895

[48] K. L. Manfreda, V. Hlebec, V. Vehovar, and U. Reja, “Open-ended vs. Close-ended Questions in Web Questionnaires The Model of Quality Aging in Place in Slovenia (QAPS) View project Contemporary Challenges of Ageing Policy in the Central and Eastern European Countries View project Open-ended vs. Close-ended Q,” Tech. Rep., 2003. [Online]. Available: https://www.researchgate.net/publication/242672718

[49] B. Freasier, G. Collins, and P. Newitt, “A Web-Based Interactive Homework Quiz and Tutorial Package to Motivate Undergraduate Chemistry Students and Improve Learning,” *Journal of Chemical Education*, vol. 80, no. 11, pp. 1344–1347, 2003.

[50] E. Kashy, B. M. Sherrill, Y. Tsai, D. Thaler, D. Weinshank, M. Engelmann, and D. J. Morrissey, “CAPA—An integrated computer[U+2010]assisted personalized assignment system,” *American Journal of Physics*, vol. 61, no. 12, pp. 1124–1130, dec 1993.

[51] J. Brunsmann, A. Homrighausen, H. W. Six, H. Six, and J. Voss, “Assignments in a virtual university–the WebAssign-System,” Tech. Rep., 1999. [Online]. Available: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Assignments+in+a+virtual+university?the+WebAssign-System{#}0

[52] P. Brusilovsky, S. Sosnovsky, and O. Shcherbinina, “QuizGuide: Increasing the Educational Value of Individualized Self-Assessment Quizzes with Adaptive Navigation Support,” pp. 1806–1813, 2004.

[53] I. H. Hsiao, S. Sosnovsky, and P. Brusilovsky, “Guiding students to the right questions: Adaptive navigation support in an E-Learning system for Java programming,” *Journal of Computer Assisted Learning*, vol. 26, no. 4, pp. 270–283, 2010.

[54] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[55] E. Freeman, E. Freeman, B. Bates, and K. Sierra, *Head First Design Patterns*. O’ Reilly amp; Associates, Inc., 2004.