Tutor for learning based on multiple choice questions

Pedro Reganha
pedro.reganha@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

December 2020

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 work 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 work 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

1. Introduction

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 the way content can be delivered, making use of the evolving network technology to deliver such contents. E-Learning is, by definition, 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.

Both Bodzin and Cates [2] and Santally and Raverdy [3] noted that in comparison to the ”traditional” approach, e-learning could take contents to the next level. It is providing their learners with more materials and different ways to look at the same topics, promoting learners with increased learning effectiveness. Having an online tool where they can train, learn and put themselves to the test is essential nowadays. This tool can have different difficulty levels, needs to be easily extended, even one simple problem could be for instance parametrised so that it is actually many more exercises than just one.

Introducing Quizzes Tutor (QT) [4, 5] a quizzing and assessment tool that allows teachers to create a multiple-choice question question bank and then use it in multiple assessment and quizzes. Its development started as an effort of project IMPRESS[1], being like so 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 upon.

QT 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 systems [6], 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.

QT 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. QT is built considering best practices and using development methodologies

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

like Domain Driven Design (DDD)[7]. 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 potent that 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 is making 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 end up dying.

2. Background

QT 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. QT is built in a 3-layered architecture:

- Presentation Layer
  - Responsible for displaying information to the user.
  - Technologies used: Vue.js[2] + Typescript[3]; Cypress[4] 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

3. Requirements

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 QT code needed to be restructured. The QT 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.

- The teacher standpoint - they require a way to easily manage questions and quizzes, similarly to what they had before. Another important point is the evaluation of the question. It currently presents a grid with the key selected for each answer, which will not suffice for more complex questions.
- The student standpoint - they needed to have a swimmingly integration with the existing questions and quizzes. Also, the answering of the new code questions needed to be intuitive.
- The developer standpoint - the system should not suffer from performance issues after these new changes. New questions should be simple to add with a low effort.

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.

4. 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.

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

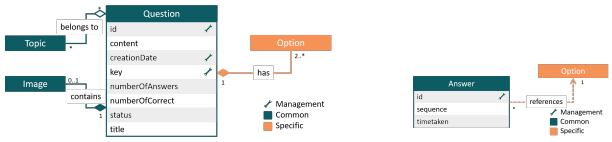
However, allowing multiple question types is a domain problem where an evaluation is required to understand the best approach.

We can identify two main focus areas when thinking about this migration/transformation of our domain: questions and question answers. These are the main ones that will be affected by the existence of multiple question types. For example, quizzes use questions the same happens for quiz answers. This means that they continue to work independently of the question type.

The question part of our domain is responsible for holding all the question information needed to manage and have the quizzes' questions. 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 solutions must have an option associated with each answer.

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. 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 and answer questions, we can see the following relevant fields, fig. 1(a) and fig. 1(b) highlights them:

- Management Fields - fields that hold the purpose of holding question metadata (e.g. `creationDate` - date of question creation);
- Common Fields - fields that are relevant for a question, however, all question types can have them (e.g. `content` - actual question text or `timeTaken` - time spent on answering the 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).

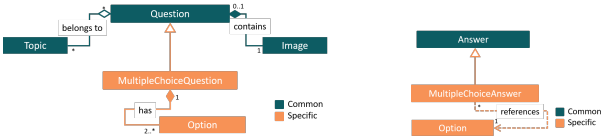


(a) Question domain model **(b)** Question answer domain model

Figure 1: Question and Question Answer domain model, highlighting specific fields

Considering these findings and the fields' separation, 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.

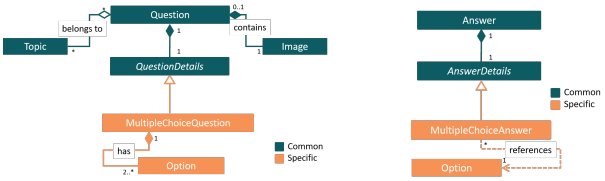
This first approach would transform the domain and create specialisations of each question and question answer. For instance, creating `MultipleChoiceQuestion` and `MultipleChoiceQuestionAnswer`, as shown in fig. 2(a) and fig. 2(b) respectively. This method was a straight forward approach. However, it 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 **(b)** QuestionAnswer transformation

Figure 2: Domain transformation of Question and QuestionAnswer using inheritance

Following the evaluation of this first approach, it was decided to proceed in a different direction. 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 [8]. 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 figs. 3(a) and 3(b), we can see the creation of the question details and the specification for each question type, in this case, only multiple choice.



(a) Question transformation **(b)** QuestionAnswer transformation

Figure 3: Domain transformation of Question and QuestionAnswer using composition

Note that, all these transformations will incur in the creation of more entities. This might impact performance, on the section 6 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.

During this stage, Frontend (FE) suffered only small changes. 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. To all relevant DTO classes was added a new field, the `type`, which maps to a specific question type. 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 **??**). For example, one of the changes was the addition of `QuestionHelpers` that manages the deserialisation on the FE side. The major difference, comparing the old with the new model, is the addition of the `questionDetailsDto` and respective `type`.

Finally, with the domain changes, it was also necessary to consider the database data's preservation. 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.

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 sec:qqt this requirement is concretised.

5. 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 User Interface (UI) — the new UI needed to accommodate both an intuitive and straightforward interface for the teachers and students.

5.1. 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 what kind of data structure would be required for constructing code questions. Figure 4 presents an example of the code belonging to an exam question. Here the student would need to identify which lines were problematic and also give a solution for them. These are the kind of problems that would require said conversion to the quiz question counterpart.

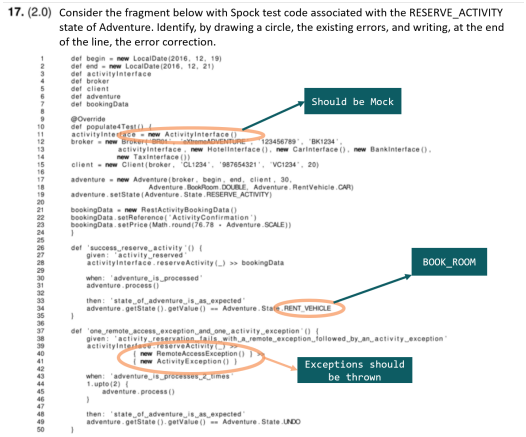


Figure 4: Sample exam question, with answers

Before starting to think technically on the solution to be implemented in QT, 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 possible problem variations and their difficulties.

1. Question - Find bug per line
   Description: Given a snippet of code identity, the lines where problems might occur. This question would 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 provide 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 provide a point.
3. Question - Fill in the blanks (Open Answer)
   Description: A quiz question similar to the previous one. The main difference is that the 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 provide 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 dissecting it, it was decided to explore the first two questions to understand the actual possibilities.

5.2. Sample implementation of exam code questions

QT had no previous code questions. Before undertaking the final solution, it was relevant to explore the problems identified in the last section (see section 5.1).

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

With these requirements we found CodeMirror[9]. 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 [5], which simplifies the usage with the FE framework.

With that in mind, we began experimenting. This experiments allowed a better validation of the selected questions and tools before integrating with the final solution. Figures 5(a) and 5(b) expose the first attempts to understand how the students would see the questions to answer and what would be required domain wise.

(a) Select lines sample question

(b) Code fill in sample question

Figure 5: First attempts of using CodeMirror to implement question answer visualization

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.

5.3. 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).

5.3.1 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 6 portraits the domain transformations done to allow for these code questions.

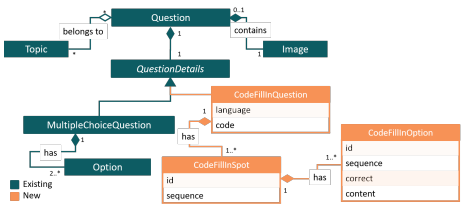
Besides these significant and more noticeable changes to the domain, it was also necessary to include a new code fill in `QuestionAnswerItem`. This domain change allows this question type to have its information logged and enables the system to perform better under large loads.

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

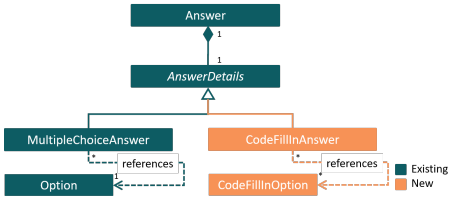
5.3.2 New backend classes and other changes

The BE 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 requiring significant

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



(a) New CodeFillInQuestion



(b) New CodeFillInAnswer

Figure 6: Question and QuestionAnswer domain model, after the introduction code fill in question

changes on that level. The only other implementation that needs to be done is the export functionality of the QT application, which allows the data to be exported to multiple formats. To qualify for a flexible code infrastructure, the code base relies heavily on the Visitor pattern[10]. This pattern provides an excellent flexible way of exporting our Questions into the multiple formats required by the application.

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 temporarily keep the answers before saving and validating its correctness with the answers package. Question's package has all the relevant classes to manage the actual questions. Also, a new question type was added to the `QuestionType` class and the `Updator` class, a visitor interface used to propagate updates, was refreshed to encompass the new question type.

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.

5.3.3 Backend changes required to add a new question

After finishing the BE transformation, it becomes even clearer the changes required to add new question types to the BE side. Figure 7 presents a schema of the developments required to update the BE 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. 7 are indeed 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

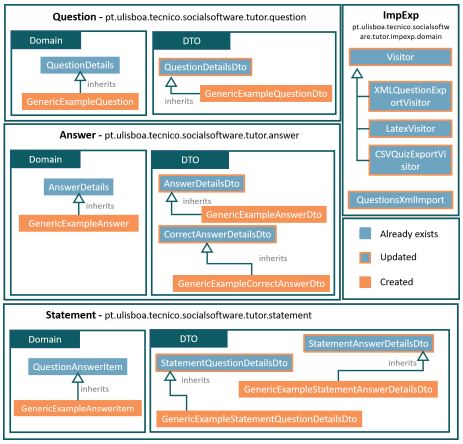


Figure 7: Backend changes required to introduce the new generic question (named: GenericExample)

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. 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.

5.4. Frontend Changes

Considering the BE tested and finished, FE was up next. As mentioned before, CodeMirror[9] was a vital part of this solution as it provided all the code editor functionalities used. As previously stated, the FE 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.

5.4.1 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 built on the BE side, we can start mapping them into Typescript.

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 deserialised from the FE side. Secondly, setup, like done for the multiple-choice questions section 4, 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 UI

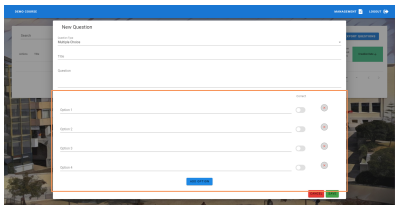
- Question Creation and Edition (`CodeFillInCreate.vue`) - Responsible for creating or editing a question. This type of component will then be used for question submission and teacher question management (see section 5.4.2)
- Question Answer (`CodeFillInAnswer.vue`) - Used by the students to answer the 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.

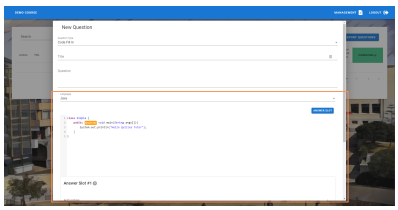
With the FE structure setup, we can continue implementing the specific UIs.

5.4.2 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`. 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.



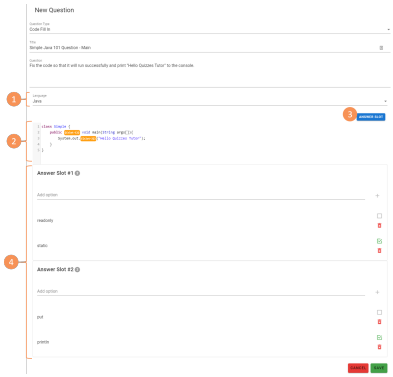
(a) Multiple Choice Question Creation Example



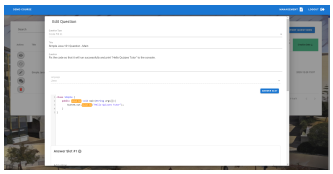
(b) Code Fill In Question Creation Example

Figure 8: New code question creation UI taking multiple question types into consideration

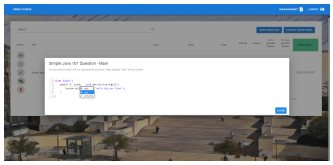
Firstly the creation/edit one, fig. 9(a) depicts the new UI. On (1), we can see the language selection box; this will determine the programming language we want our code to be. On (2), we have our code editor, which will be highlighted with the correct programming language. (3) shows the button that 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. Note that fig. 9(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.



(a) Create UI with some highlights.



(b) Edit UI



(c) View UI

Figure 9: Code Fill In Question Management

Secondly, the visualization one, fig. 9(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 FE, 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.

5.4.3 Students' interface to answer and validate it

It was necessary to add the question to the students' quiz visualisation UI. To ensure that the User Experience (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. 10), 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 11 presents the UIs implemented for this new question type. This red-box basically represents the answer (`CodeFillInAnswer.vue`) and answer response (`CodeFillInAnswerResponse.vue`) components.

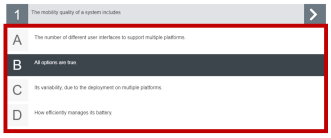


Figure 10: Multiple choice question, with the `MultipleChoiceAnswer.vue` component identified in red

The final result of this implementation is depicted in fig. 11. Note that it draws inspiration from the initial samples done (see section 5.2 sub-section). This UI allows for an intuitive interaction by the learners.

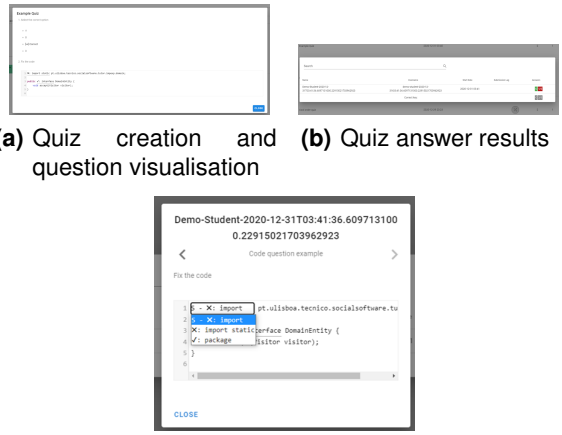
(a) Code fill in question example **(b)** Code fill in answer example

Figure 11: Code fill in question type used in quiz

5.4.4 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 5.4.2 was developed. The component used here is the `CodeFillInView.vue`.

Figure 12 portraits the multiple interfaces used to manage quiz questions and how it is affected by this code question. Figure 12(a) shows how all questions can be viewed before creating the quiz. Whilst, Figures 12(b) and 12(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. 12(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. 12(c)) allows the teacher to explore each of the errors shown in more detail.



(a) Quiz creation and question visualisation **(b)** Quiz answer results

(c) Quiz answer details

Figure 12: Code fill in question type used in quiz

5.4.5 Frontend changes required to add a new question

As noted from the previous sub-chapters, the FE 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 13 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.

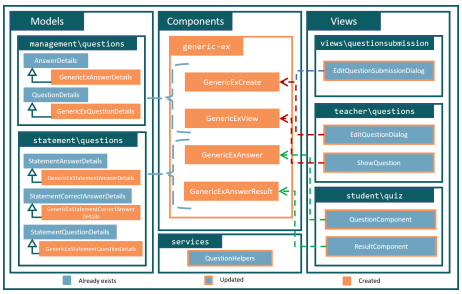


Figure 13: Frontend changes required to introduce the new generic question (named: GenericEx)

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

6. Evaluation

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

6.1. Performance Evaluation

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[6] (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.

The tests ran where the following:

- Question Creation - consists of login in as a teacher or multiple teachers of a given course and creating a given number of questions.
  - Fifty teachers to fifty questions (with teardown)
  - Fifty teachers to fifty questions (without teardown)
- Quiz interaction - 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 QT, the quizzes.
  - 100 students answer 20 questions
  - 300 students answer 20 questions
  - 300 students answer 40 questions

The results of the tests conclusively showed that when there where variations were relatively small. Tables 1 and 2 present the data of part of the last test executed, which was the most extensive and represents well the results obtained across all trials. As we can see, there is a small (100 milliseconds) slowness on the new implementation. This result was actually shifted in other tests. The third step, which considers all the answers provided, currently in a logged state, and saves them properly in the database, showed quite problematic results, both for the new and old solution. However, this result showed worse performance in the new solution, which 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.

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

Table 1: Results: Students answering the quiz simulation (40 Questions + 300 Students) - Original implementation

Label	Samples	Average	Median	Min	Max	Throughput
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
TOTAL	13200	1728	548	10	46272	167,49994

Table 2: Results: Students answering the quiz simulation (40 Questions + 300 Students) - Multiple Question Types implementation

Label	Samples	Average	Median	Min	Max	Throughput
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
TOTAL	13200	1879	569	10	52026	155,10981

6.2. Flexibility Evaluation

One of the requirements previously stated (see section 3) 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 FE (section 5.4.5) and the BE (section 5.3.3), 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[11].
- 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.

With the new implementation finished, it was time to assess it. The BE changes updated 28 files(16 creations and 12 modifications), 977 insertions lines of code, 22 deletions lines of code. The developments took 2 hours, 1h35 for the initial development, and the remainder of the time spent with fixes while developing the FE. The FE changes updated 22 files(14 creations and 8 modifications), 815 insertions lines of code, 6 deletions lines of code. The developments took 7 hours, which is considerably more than the BE part and can be easily explained by the fact that this development requires more creative sides.

Although it might seem lots of changes, the updates, for instance, are very simple in all files, and the new creations are mostly implementing boilerplate code. Looking at the data, both files changed and time taken, this solution is up to the challenge of adding 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 FE.

7. Conclusions

Throughout the years, researchers have been exploring the best tools and methodologies to ensure that, as a teacher, the message passes down successfully. 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 practical tools that can help in this effort. Throughout the research explored during this thesis, we can see multiple engaging devices.

This work 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 4). This transformation allows for the platform to continue its growth with ease. It is important to note that there where some hindered performance in a particular test. Still, it is not enough to discredit the current solution.

This thesis's main focus was unquestionably adding a new programming code question to the Quizzes Tutor platform. During this work, there was an evaluation of multiple systems. Still, 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. This analysis is, without a doubt, relevant to be explored in future work.

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—the new question type and a system that can be easily extended with new types.

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. 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.

References

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

[2] 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.

[3] 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.

[4] "Quizzes Tutor Website." [Online]. Available: https://quizzes-tutor.tecnico.ulisboa.pt/

[5] "Quizzes Tutor Source Code." [Online]. Available: https://github.com/socialsoftware/quizzes-tutor

[6] 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.

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

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

[9] "CodeMirror." [Online]. Available: https://codemirror.net/

[10] 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.

[11] 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