

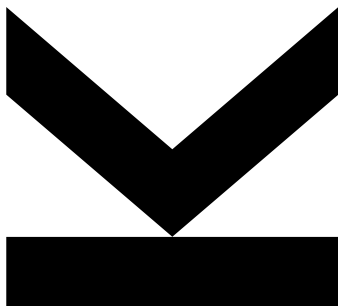
Author
Rebecca Rachinger

Submission
**Institute for System
Software**

Thesis Supervisor
**Dipl.-Ing. Dr. Markus
Weninger**

March 2023

Fill-in-the-blanks Questions for the Online Exam System Xaminer



Bachelor's Thesis
to confer the academic degree of
Bachelor of Science
in the Bachelor's Program
Informatik

Bachelor's Thesis

Fill-in-the-blanks Questions for the Online Exam System Xaminer

Student: Rebecca Rachinger

Advisor: Dipl.-Ing. Dr. Markus Weninger, BSc

Start date: March 2022

Dipl.-Ing. Dr.

Markus Weninger, BSc

Institute for System Software

P +43-732-2468-4361

F +43-732-2468-4345

markus.weninger@jku.at

The exam system Xaminer is used by the Institute for System Software and other institutes to provide online exams for students. To improve the experience for both, lecturers and students, the system is constantly being improved and extended. Especially fill-in-the-blanks questions are currently missing.

The goal of this thesis is to introduce this question type to the system. The student has to develop convenient editing features to allow lecturers to easily define fill-in-the-blanks questions. For this, lecturers should be able to provide a reference solution to a given question and then should be able to select those parts of the solution that should be displayed as blanks during the exam.

Besides blank text areas that have to be filled out textually by the exam taker, lecturers should also be able to define "dropdown blanks". For this kind of blank the lecturer provides a number of possible solutions, out of which one has to be chosen from a dropdown input element by the exam taker.

Modalities:

The progress of the project should be discussed at least every three weeks with the advisor. A time schedule and a milestone plan must be set up within the first 3 weeks and discussed with the advisor. It should be continuously refined and monitored to make sure that the thesis will be completed in time. The final version of the thesis must be submitted not later than 31.09.2022.

Abstract

Xaminer is an online exam tool developed by the Institute for System Software at the Johannes Kepler University in Linz. However, an important question type that is often used in other tools but is missing in Xaminer is the fill-in-the-blanks question type. In a fill-in-the-blanks question students must write the missing text in the blank. This thesis aims to integrate this kind of question into Xaminer. To create a fill-in-the-blanks question, the lecturer enters the question into a text field and uses buttons to convert text parts to blanks. There are three types of blanks: (1) text blanks, where students have to enter a single word or even whole sentences, (2) numerical blanks, which require students to enter a numeric value, and (3) dropdown blanks, where the student has to choose one option from several given options. It is important to note that a single fill-in-the-blanks question can consist of several different types of blanks. Furthermore, in a graphical editor, the solution for the blank can be edited.

Kurzfassung

Xaminer ist ein Online-Klausuren Tool, welches vom Institut für Systemsoftware an der Johannes Kepler Universität in Linz entwickelt wurde. Eine wichtige Frageart, welche fehlt, aber oft in anderen Tools verwendet wird, ist die Lückentext-Frage. Bei einer Lückentext-Frage müssen die Studierenden den fehlenden Text in die Lücke schreiben. Das Ziel dieser Arbeit ist es, diese Frageart in Xaminer zu integrieren. Um eine Lückentextfrage zu erstellen, gibt der Dozent oder die Dozentin die Frage in ein Textfeld ein und wandelt mithilfe von Schaltflächen Abschnitte des Textes in Lücken um. Es gibt drei Arten von Lücken: (1) Textlücken, bei denen der Student oder die Studentin ein einzelnes Wort oder sogar ganzen Sätze eingeben muss, (2) numerische Lücken, bei denen Studierende einen numerischen Wert eingeben müssen, und (3) Dropdown Lücken, bei denen der Student oder die Studentin eine Option aus mehreren vorgegebenen Optionen auswählen muss. Es ist wichtig zu beachten, dass eine einzelne Lückentextfrage aus mehreren verschiedenen Lückentypen bestehen kann. Außerdem kann in einem grafischen Editor die Lösung für die Lücke bearbeitet werden.

Table of Content

Contents

1	Introduction	1
2	Background	3
2.1	Xaminer	3
2.1.1	Create Exams	3
2.1.2	Manage Student Groups and Supervisors	3
2.1.3	Correct Exams	4
3	Idea	5
3.1	General Idea	5
3.2	First Approach: Keywords	5
3.3	Second Approach: Graphical Editor	6
4	Realization	9
4.1	General Part	10
4.2	Creating a Fill-in-the-blanks Question	10
4.3	Blank Editor	12
4.4	Visualizing the Question	15
5	Usage	19
6	Related Work	25
7	Current Limitations and Future Work	27
8	Conclusion	29
	Literature	31

1 Introduction

As a result of the coronavirus pandemic face-to-face teaching at the university was no longer possible. Within a short time, lecturers had to switch to online teaching. Online learning platforms such as Moodle have been used to support lecturers in interacting with students. By using Moodle, lecturers can provide students with documents, upload teaching videos and assignments, or create exams. However, some features were missing when designing exams via Moodle such as syntax highlighting for programming questions or drawing pictures. Therefore, the Institute for System Software at the Johannes Kepler University in Linz developed an online exam tool called Xaminer. Xaminer can be used to create and hold exams. It is also possible to send emails to students and supervisors with streaming links and exam links. After the exam, lecturers can grade the submitted exams. The exam tool is constantly being developed to provide convenience to both students and teachers.

Additionally, an important question type that is missing in Xaminer is the fill-in-the-blanks question type. A fill-in-the-blanks question consists of sentences or paragraphs containing blanks that must be completed by the student by typing words or phrases into the blank or selecting the correct option from a dropdown list. The purpose of this thesis was to implement this kind of question. To create such a question, three different types of blanks can be used: (1) text blanks, (2) numerical blanks, and (3) dropdown blanks. A fill-in-the-blanks question can be easily created by entering a *reference solution* which is the full fill-in-the-blanks text including the solutions of the blanks into the reference solution text field. Afterward, we select the solution in the reference solution with the mouse and convert the selected solution into a blank by clicking the corresponding blank type button. Each blank type (text blank, numerical blank and dropdown blank) has its own button to convert the selected text into a blank. Furthermore, the solution for the blank can be edited in a graphical editor called *blank editor*, which appears automatically under the reference solution text field after we clicked the blank type button. If we need another blank, the same process as previously described can be repeated. Special attention has been paid to the easy and user-friendly creation of the question. In a preview, the lecturer sees how the fill-in-the-blank question will look like for the students.

2 Background

This section provides an overview of the tool Xaminer and explains some of its most essential features.

2.1 Xaminer

Xaminer is an online exam tool that is developed by the Institute for System Software at the Johannes Kepler University in Linz. It has features such as syntax highlighting, spacing with a tab or drawing a picture that other online tools such as Moodle do not provide. These features are especially important for programming exams and make it comfortable for the students. With Xaminer we can (1) create exams, (2) manage student groups and supervisors, and (3) correct exams. Now we take a closer look at these features.

2.1.1 Create Exams

When we create a new exam we must first set the exam title, the exam date, the start time and the end time. An exam can consist of several *exam blocks*. An exam block is essentially an exam question or some plain text. At the moment we can choose between these five *question types*:

1. **Text question:** The student has to answer by writing a free text.
2. **Code question:** The student has to answer by writing source code, e.g. a function or a class.
3. **Drawing question:** The student has to answer by drawing a picture, e.g. drawing a class diagram.
4. **Single choice question:** The student has to choose one answer of multiple possible answers.
5. **Multiple choice question:** The student can choose more than one answer of multiple possible answers.

2.1.2 Manage Student Groups and Supervisors

After we create an exam we can add student groups and supervisors to this exam. This has the significant advantage that every student and supervisor automatically gets an email with exam information, a streaming link and an exam link. For this we must first

specify how many student groups we have. Then we assign a supervisor and a streaming link to each student group. After that the students must be added to these groups. We can do this either by uploading a CSV file or by adding them manually by entering each student ID, study code, first name, last name and email in a text fields. Finally, there are two predefined email texts one for the students and one for the supervisors. Both texts contain information about the exam, the streaming link and the exam link. The predefined texts can be adjusted as required. Xaminar provides two buttons, one to send the email to the supervisors and one to send the email to the students.

2.1.3 Correct Exams

After the exam the submitted exams can be exported as TXT, HTML or PDF. We will receive a TXT, HTML or PDF file for each submitted exam. Single choice questions and multiple choice questions are corrected automatically. All other questions (text questions, draw questions and code questions) must be manually corrected. The exam submissions in HTML and PDF format are formatted nicely which makes correcting the exam questions very easy.

3 Idea

This section discusses the general idea of how a fill-in-the-blanks question type should be designed and presents two approaches on how to implement this question type. Furthermore, the advantages and disadvantages of each solution are discussed. Finally, it is explained which approach we have chosen and why.

3.1 General Idea

The basic idea is that we enter a *reference solution*, which is the full fill-in-the-blanks text including the solutions of the blanks, into the reference solution text field and mark (e.g. with keywords) each part of the text that should be a blank. The student must then write the correct answer in the blank. There are three *blank types* we want to support:

1. **Text blanks:** The student must write a free text in the blank.
2. **Numerical blanks:** The student must write a number into the blank.
3. **Dropdown blanks:** The student must choose one answer of multiple possible answers.

A fill-in-the-blanks question can contain multiple blanks of different blank types.

3.2 First Approach: Keywords

Our first approach was to use keywords to designate a blank. For this, we have to mark the start and the end of the text which should be turned into a blank with a keyword. We chose for the keyword $\$$, e.g. $\$int\$$ $age = 24;$ means that int should be converted to a blank. To distinguish between the different blank types we decided to include a letter in the first keyword section, we took f for a text blank, n for a numerical blank, and d for a dropdown blank. For the dropdown blank we added a second keyword $;;$ to allow multiple answer choices. For example $int;;string;;boolean$ means that the student can choose between three possible answers: int , $string$ and $boolean$. Here is an example of how a blank of each blank type can look:

- text blank: $\$fint\$$ $age = 24;$
- numerical blank: int $age = \$n24\$;$
- dropdown blank: $\$dint;;string;;boolean\$$ $age = 24;$

A big advantage of this approach is that implementing this question requires little effort. We just need a text field to write the question and mark the blanks with keywords. Another benefit is that the solution for the blank is directly in the reference solution so we can edit the solution right there. Also extending the blank types or adding a new blank type is very easy as we just need a different or additional keyword. This leads us to a major disadvantage which is that keywords require the user to learn a new syntax. We need to know the meaning of all keywords and also know where to put them. The keywords can also make the reference solution messy or bloated. In addition, it is difficult to recognize the solution in the reference solution at first glance, since the solution differs from the reference solution only in the keyword at the beginning and end. This happens especially when the solution consists of several lines. In addition, many people are unfamiliar with the use of keywords, which can make people reluctant to use them.

3.3 Second Approach: Graphical Editor

Our second approach uses a text field for the reference solution, similar to the first approach. However, the blanks are added via buttons. There is a button for each blank type. To create a blank, we have to click the respective blank type button. This appends a *keyword string* to the reference solution at the current cursor position. For example, the keyword string $\$TEXT-1$ is inserted at the current cursor position when we press the corresponding text blank button. If we now add a second text blank by clicking the text blank button again, the keyword $\$TEXT-2$ is appended to the reference solution. In addition, it is also possible to select the solution in the reference solution and then click on the desired blank type button. Therefore, the selected solution is replaced by a keyword string, e.g. $\$TEXT-3$ if we clicked on the text blank button. Furthermore, the selected solution is inserted in a *blank editor*, which was added automatically after the reference solution text field. A blank editor is a graphical editor where we can edit the solution. Every blank has its own blank editor. If we had added a blank by only clicking on a blank type button, the solution in the blank editor is empty. More detailed information is provided in Section 4.

The big benefit of this approach is that creating a fill-in-the-blanks question is very easy, as the buttons and the blank editor are self-explanatory. Another positive aspect is that it can be easily expanded without complicating the question creation. However, a problem that can occur is that it can become confusing when we have many blanks in a question because of the many blank editors.

The second approach was chosen because creating the fill-in-the-blanks question is easier with the help of a graphical editor. Another reason why the second approach was chosen is that expanding the fill-in-the-blanks question does not make the question creation more complicated for the user.

4 Realization

This section shows how a fill-in-the-blanks question type is structured and what kind of features it has. First, we will discuss the general part where we can provide information for the students about the question. Then we talk about how to create a question and how to edit a solution of a blank in a blank editor. Finally, we will show what the question look like in the preview and exam look like.

In Figure 1 we see an example of a fill-in-the-blanks question on the *exam main page*. The exam main page is the page where we create the exam. It is divided into two parts: on the left, we create and edit the question ((1), (2) and (3)) and on the right, we see the question preview ((4)).

(1) is the general part where we can provide information for the students, e.g. information about how to solve this question. (2) is the part where we we add and edit the reference solution. (3) shows the blank editor and the solution *? extends E* for the text blank. (4) is the preview of the fill-in-the-blanks question and is constantly updated while editing the question.

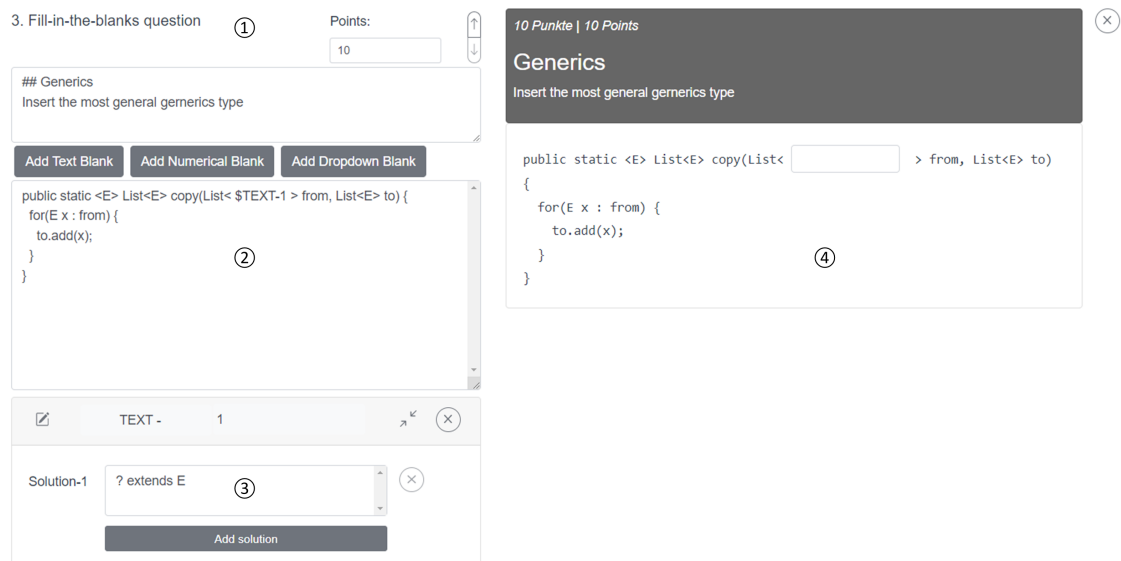
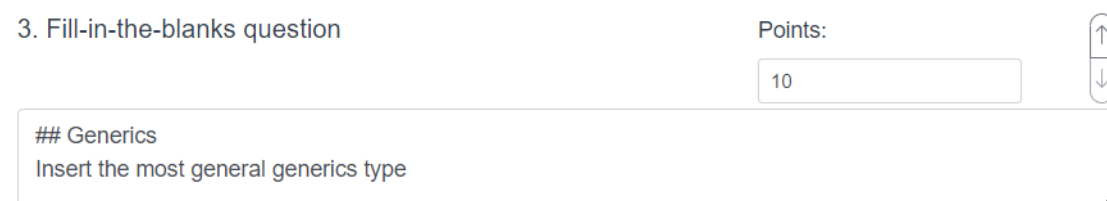


Figure 1: Overview of an example of a fill-in-the-blanks question. On the left side we create the question and on the right side we see the preview of the question.

4.1 General Part

In the general part (see Figure 2) we give information about the question, e.g. the achievable points for the question. With two arrows we can change the order of the exam questions. In the task description text field we set the question's title and instructions in Markdown [2]. Markdown is a lightweight markup language that enables users to format text using a simple syntax. The general part is the same for all question types and has already been implemented.



3. Fill-in-the-blanks question

Points: 10

Generics
Insert the most general generics type

Figure 2: In the general part we provide information for the students.

4.2 Creating a Fill-in-the-blanks Question

We can create a fill-in-the-blanks question with the reference solution text field and three buttons for each blank type shown in Figure 3. If we need a blank we have to click on the corresponding button. After clicking, a keyword string is added at the current cursor position and a blank editor is added underneath the reference solution text field. The keyword string is basically the blank's name and consists of two parts which are separated by a hyphen. The first part is the *blank type keyword*. The blank type keyword depends on the blank type and consists of \$ and the uppercase *blank type*. Following blank type keywords exist in Xaminer: (1) \$TEXT. (2) \$NUMERICAL and (3) \$DROPDOWN. The second part of the keyword string is the *blank name*. It is generated automatically and can be edited later in the blank editor (see Section 4.3). The blank name is necessary so that each blank can be assigned to the corresponding blank editor.

There are three possible ways to add a blank:

1. We click on a blank button when no text is selected: We get a blank editor where the blank's solution is empty.
2. We select a text and then click a blank button: We retrieve a blank editor where the blank's solution is the selected text.
3. We manually write the keyword and the blank name into the text field: We get a blank editor where the solution is empty.



Figure 3: With the reference solution text field and the three buttons we can create a fill-in-the-blanks question.

4.3 Blank Editor

In the blank editor (see Figure 4) we can edit the blank and specify its solution. Each blank type has its own blank editor which can be uniquely assigned by the blank name. The blank editor appears automatically as soon as a new keyword string is added to the reference solution text field. The blank editor consists of *header* and *body*. The header is the same for each blank type and the body is different for each blank type.

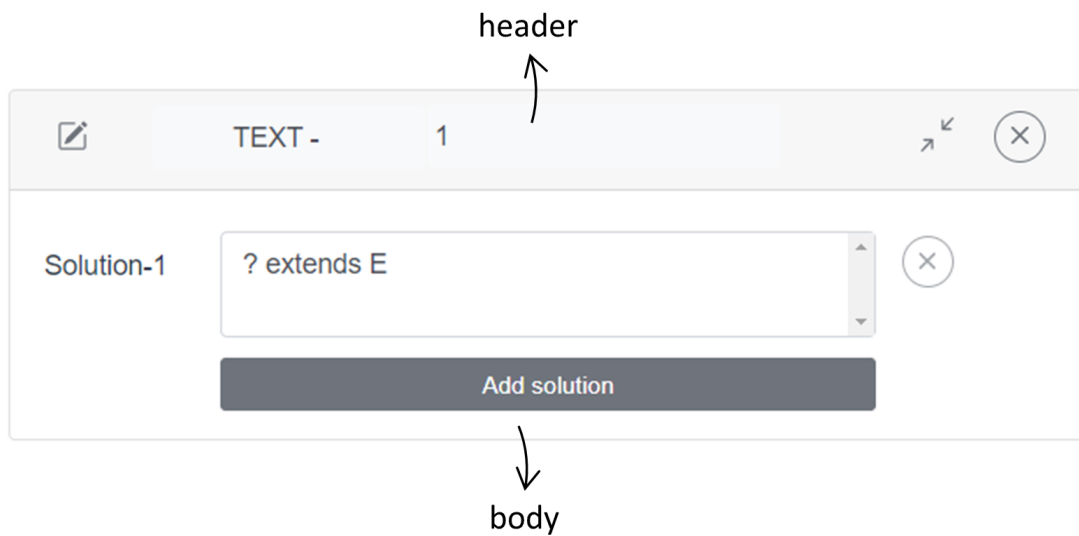


Figure 4: We can edit the blank's solution in the blank editor.

Header

The header (shown in Figure 5) is the same for every blank type. ② shows the keyword string composed of the blank type and the blank name. We can also minimize or maximize the blank editor (③). If we minimize the blank editor the body disappears and we only see the header. With ④ we can delete the blank. Both the blank editor and the keyword string in the reference solution text field disappear.

By clicking the edit icon (①) the header will change and we can edit the blank type and the blank name as shown in Figure 6. While the header is in edit mode the body is greyed out and can not be edited. We can change the blank type via a dropdown menu that lists all blank types. The blank name can be edited in the blank name text field next to the dropdown. The blank name must not be empty, consist of one word and not already exist. If at least one of these requirements is not met the blank name text

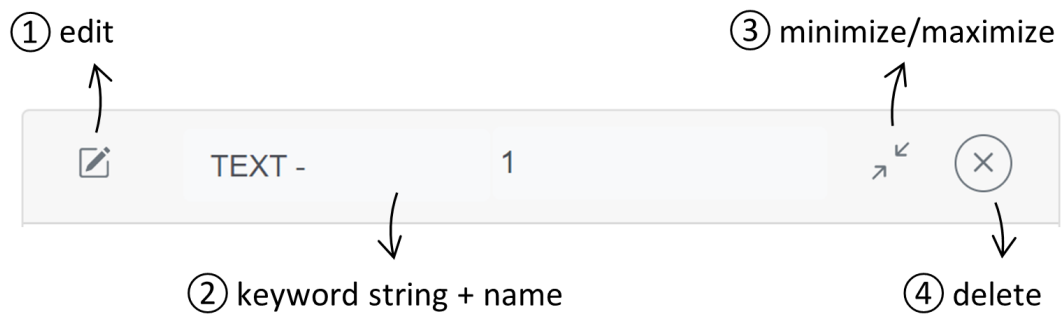


Figure 5: Header of the blank editor. Every blank type has the same header.

field gets a red border and a warning message appears below the blank name text field. Changing the blank type or the blank name keeps the solution in the solution text field. To make the changes effective we must click on the blue save button. Then we get the header as in the beginning and we can edit the body again.

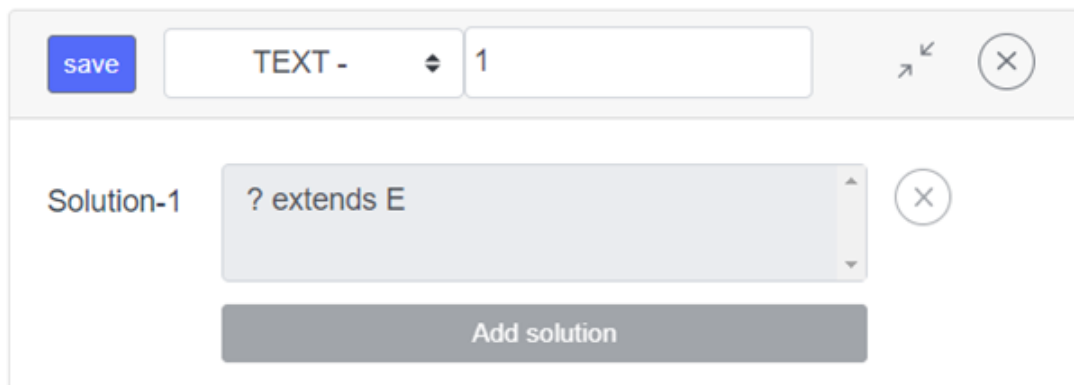


Figure 6: After clicking the edit icon we can edit the blank type and the blank name.

Body

In the body, we specify the solution for text blanks and numerical blanks or the choices of dropdown blanks. Consequently, the body of the blank editor is slightly different for each blank type.

1. Text blank

In Figure 7 we see the body of a text blank. In the solution text field, we can enter the solution for the blank. The solution can consist of a single word or even entire sentences. It is possible to enter additional solutions by clicking the *Add solution* button. An additional solution is needed when for example the student has to define a weight field in a *Person* class because the correct solution would be *float weight;* or *double weight;*. After clicking the *Add solution* button, a second solution option consisting of the solution's identifier (e.g. Solution-2), a solution text field and a delete button appears below the first solution. We can use the delete button (circle with an X inside) to delete the blank. Deletion is only possible if there are at least two solutions.

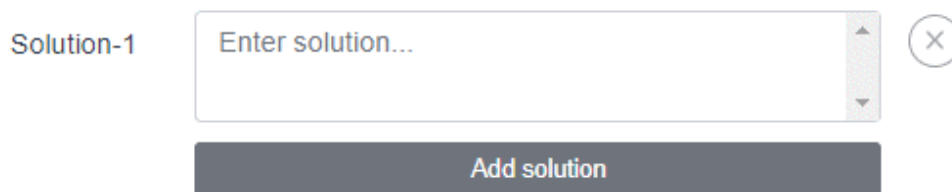


Figure 7: Body of a text blank's editor.

2. Numerical blank

Figure 8 shows the body of a numerical blank. It has the same format as a text blank's editor body except that we can only enter a positive or negative number as solution. We can use either a comma or a dot as floating point. If the entered solution is not a number as defined above, the border of the solution text field will turn red and a warning message will appear below the solution text field.

Figure 8: Body of a numerical blank.

3. Dropdown blank

In the body of a dropdown blank (shown in Figure 9) we specify the different choices by clicking the *Add choice* button and writing the choices into the solution text field. We use the radio buttons to indicate the correct solution for the dropdown blank. Only one radio button can be selected. We can remove a choice by clicking the delete button.

Figure 9: Body of a dropdown blank's editor.

4.4 Visualizing the Question

We visualize the fill-in-the-blanks question on the main exam page, where the lecturer creates the exam and on the exam page, where the students write the exam. In order to visualize the fill-in-the-blanks question, we need to first transform the reference solution into a suitable format. Once we have done that, we can easily visualize the question. This is accomplished with the help of two frameworks: Vue.js [4] and BootstrapVue [1]. Vue.js provides the tools to create the interactive elements of the question, while BootstrapVue handles the formatting and presentation of the question.

Data Transformation

First, we need to split the reference solution based on the blank type keywords, which is accomplished through the use of a regular expression. Consequently, we get an array in which each entry is either a text or a keyword string. For example, if the reference solution is *List<Integer> list= \$TEXT-1 ;*, the array after splitting would have three entries and

would look like this: `["List<Integer> list= ", "$TEXT-1", " ;"]`. Next, this array is mapped to another array called `questionParts`, which has the type `FITBQuestionParts`. This enables us to store relevant information such as the blank type, the solutions, and the answer of the student. Additionally, we can store the width (class field `cols`) and the height (class field `rows`) of the blank in the `FITBQuestionParts` class. This is essential because the blank should have the same width as the longest solution (Figure 10, (2a), (2b)). In cases where the solution has more than one line, the width should be the maximal width and the blank should have the same height as the solution (Figure 10, (1a), (1b)).

The screenshot shows a quiz interface for a "2. Fill-in-the-blanks question" worth 30 points. The question asks to complete an "Unsorted List" class. The code editor shows the following code:

```
class List {
    Node head = null;

    // inserts the value at the beginning of the list
    void prepend(int val) {
        $TEXT-1
    }

    boolean contains(int val) {
        Node p = head;
        while ( $TEXT-2 ) {
            p = p.next;
        }
        // p == null || p.val = val
        return p != null;
    }
}
```

Below the code editor are two solution input fields:

- Solution-1:** A text field containing `Node p = new Node(val); p.next = head; head = p;` with a width indicator (1a).
- Solution-2:** A text field containing `p != null && p.val != val` with a width indicator (2a).

On the right, a dark grey box titled "Unsorted List" shows the same code with two input fields:

- (1b):** A wide text field for the `prepend` method.
- (2b):** A text field for the `contains` method's while loop condition.

Figure 10: The blank text field is as wide as the solution. When the solution has more than one line then the blank text field has the maximal width.

Visualization

After converting the data into the desired format, the next step is to visualize the fill-in-the-blanks question. To achieve this we used the two frameworks Vue.js [4] and BootstrapVue [1]. Using *v-for* provided by Vue.js, we create a span element for each entry in the *questionParts* array, as shown in Figure 11, (1). In order to distinguish between the different blank types and display the appropriate blank element, we used the directive *v-if* from Vue.js, which only displays the HTML element if the directive's expression returns true. We have five different HTML elements that should be visualized, depending on the blank type. We differentiate between text blanks where the solution has only one row (Figure 11, (2)) and more than one row (Figure 11, (3)) because the text field should have the maximal width when the solution has more than one row. When it is a numerical blank (Figure 11, (4)), we also display a text field, but only numbers are allowed to be entered. If the entered text is not a number, the border of the blank text field becomes red. A select element (Figure 11, (5)) where we can choose one choice of several choices is used if it is a dropdown blank. If it is not a blank, we display the text in a span element (Figure 11, (6)). An example of the preview and the exam is in Section 5.

```

<span
  v-for="(blankPart, blankPartIndex) in blankTextQuestion.questionParts"
  :key=" 'BLANK_TEXT_QUESTION_answer_' + blankPartIndex"
  :class="blankPart.kind=== 'NONE' ? 'd-inline align-text-bottom' :
    (blankPart.kind === 'TEXT' && blankPart.rows > 1) ? 'w-75 d-inline-block' : 'd-inline-block align-bottom'"
  <!-- TEXT blank single line -->
  <textarea
    v-if=" blankPart.kind=== 'TEXT' && blankPart.rows === 1"
    :cols="blankPart.cols"
    :rows="blankPart.rows"
    v-model="blankPart.answer" style="width: auto; height: 34px;" class="form-control blank-text-question-textarea"/>
  <!-- TEXT blank multiple line -->
  <textarea
    v-if="blankPart.kind === 'TEXT' && blankPart.rows > 1"
    :rows="blankPart.rows"
    v-model="blankPart.answer" class="form-control blank-text-question-textarea" style="width: 100%; height: auto;" />
  <!-- Numerical -->
  <textarea
    v-if="blankPart.kind === 'NUMERICAL'"
    :cols="blankPart.cols"
    :rows="blankPart.rows"
    v-model="blankPart.answer" style="width: auto; height: 34px;"
    :class="blankTextQuestion.isNumberOrEmpty(blankPart.answer)? 'form-control blank-text-question-textarea':
      'form-control is-invalid blank-text-question-textarea'" />
  <!-- Dropdown -->
  <b-form-select
    v-if="blankPart.kind === 'DROPDOWN'"
    v-model="blankPart.answer" plain class="w-auto" style="min-width: 100px; display: inline-block;"
    <b-form-select-option
      v-for="(choicePart, choiceIndex) in blankPart.dropDown.choices"
      :key=" 'BLANK_TEXT_QUESTION_choice_answer_' + choiceIndex" :value="choicePart">
        {{ choicePart }}
    </b-form-select-option>
  </b-form-select>
  <!-- Text -->
  <span
    v-if="blankPart.kind === 'NONE'"
    style="white-space: pre-wrap; line-height: 1.6">{{ blankPart.text }}</span>
</span>

```

Figure 11: We display a blank text field for both text and numerical blanks, while a select element is used for dropdown blanks.

5 Usage

In this section we will use an example to show how to create a fill-in-the-blanks question in Xaminer.

After logging into Xaminer we are on the main page where we can see upcoming and past exams. To create a new exam we have to click on the *New* button shown in Figure 12.

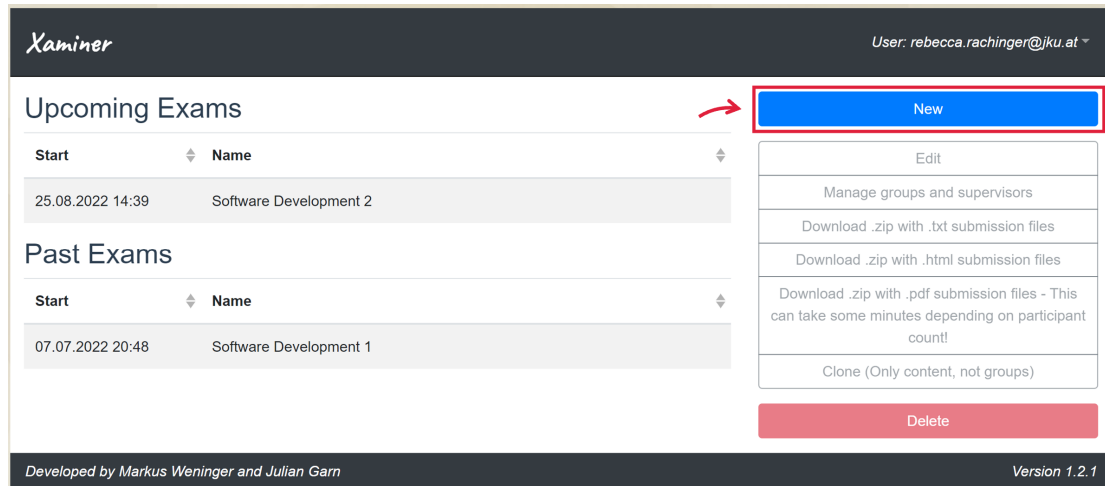


Figure 12: On the main page we see upcoming and past exams. We can also add and delete exams on this page.

Clicking the button forwards us to the exam main page, which is displayed in Figure 13. Here we set the title, date, start time and end time of the exam. Additionally, we can provide general exam information for the students in the first text block. After we have completed these initial steps, we can move on to adding a fill-in-the-blanks question. To do this, we simply need to select the fill-in-the-blanks question type from the question types dropdown and then press the *Add* button, which is also shown in Figure 13. Doing so inserts a fill-in-the-blanks question block after the exam block *Text block*, as shown in Figure 14. We set the points to 20 (Figure 14, ①) and enter the text *Complete the Student class* in the task description text field (Figure 14, ②). Next, we insert the reference solution into the reference solution text field. We want ③ to be a dropdown blank, ④ and ⑤ to be a text blank, and ⑥ to be a numerical blank.

To convert ③ to a dropdown blank we need to select the word *implements* with the cursor and click the *Add Dropdown Blank* button. This brings up a blank editor for the dropdown blank under the reference solution text field (Figure 15, ③a). On

Xaminer User: rebecca.rachinger@ku.at

Exam main page

Exam name	Date	Start time	End time
Creating a fill-in-the-blanks question	25.08.2022	14:50	15:50

Exam blocks

(text written in Markdown)

1. Text block

Points:

```
## Fill-in-the-blanks question

This exam shows how to create a fill-in-the-blanks question.
```

Preview

Fill-in-the-blanks question

This exam shows how to create a fill-in-the-blanks question.

Fill-in-the-blanks
Add

Markdown Help

Status

Overall points (calculated): 0

Full exam preview
Go back to overview Save

Exam not yet started, edit allowed

Developed by Markus Wening and Julian Gam Version 1.2.1

Figure 13: On the exam main page we define the exam content by adding exam blocks. An exam block is a text block or an exam question. The question type can be selected from the highlighted exam types dropdown.

the preview *implements* has been replaced with a dropdown (Figure 15, ③b). In the dropdown blank, we want to be able to choose between *implements* and *extends*. Since we selected the word *implements* it was automatically added as a choice in the dropdown blank editor. We add the choice *extends* by using the plus button in the dropdown blank editor. We do not need to change the radio button because in our case the correct answer is already selected. The dropdown blank is now done and we convert ④ and ⑤ into a text blank. We select *this.name = name; this.age = age;* and click on the *Add Text Blank* button. This adds another blank editor for the text blank after the dropdown blank editor (Figure 15, ④a). The text in the preview is converted to a two-line text field since our solution also has two lines (Figure 15, ④b). We add a second solution by clicking the plus button in the blank editor and entering the solution *this.age = age; this.name = name;* in the solution text field. After that, we similarly convert ⑤ into a text blank (Figure 15, ⑤a and ⑤b). Now we need to transfer ⑥ into a numerical blank by selecting the text and clicking the *Add Numerical Blank* button (Figure 15, ⑥a and ⑥b).

We change the blank name of all blanks in each blank editor by clicking on the edit button (square with a pen in it), writing the new blank name in the blank name text field and pressing the save button. Figure 15 shows the final fill-in-the-blanks question. In Figure 16 we see how the question looks like during an exam taken by students.

2. Fill-in-the-blanks question

Points: ①

20

Comparable ②

Complete the Student class

Add Free Choice

Add Numerical

Add Dropdown

```
public Student implements Comparable<Student> {  
    // fields  
    String name; ③  
    int age;  
  
    // constructor  
    public Student(String name, int age) {  
        this.name = name; ④  
        this.age = age;  
    }  
  
    @Override ⑤  
    public int compareTo(Student other) {  
        int comp = Integer.compare(age, other.age);  
        if (comp == 0) { ⑥  
            comp = name.compareTo(other.name);  
        }  
        return comp;  
    }  
}
```

20 Punkte | 20 Points

Comparable

Complete the Student class

```
public Student implements Comparable<Student> {  
    // fields  
    String name;  
    int age;  
  
    // constructor  
    public Student(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    @Override  
    public int compareTo(Student other) {  
        int comp = Integer.compare(age, other.age);  
        if (comp == 0) {  
            comp = name.compareTo(other.name);  
        }  
        return comp;  
    }  
}
```

Figure 14: A fill-in-the-blanks question with a sample question where the red marked words have to be transformed into blanks.

2. Fill-in-the-blanks question Points:

Comparable
Complete the Student class

```

public Student $DROPDOWN-interface Comparable<Student> {
    // fields
    String name;
    int age;

    // constructor
    public Student(String name, int age) {
        $TEXT-constructor
    }

    $TEXT-override
    public int compareTo(Student other) {
        int comp = Integer.compare(age, other.age);
        if(comp == $NUMERICAL-compareTo) {
            comp = name.compareTo(other.name);
        }
        return comp;
    }
}

```

20 Punkte | 20 Points

Comparable
Complete the Student class

```

public Student  Comparable<Student> {
    // fields
    String name;
    int age;

    // constructor
    public Student(String name, int age) {
        
    }

    
    public int compareTo(Student other) {
        int comp = Integer.compare(age, other.age);
        if(comp == ) {
            comp = name.compareTo(other.name);
        }
        return comp;
    }
}

```

3a DROPDOWN - interface

implements

extends

1 Mark correct answer

4a TEXT - constructor

Solution-1

Solution-2

5a TEXT - override

Solution-1

6a NUMERICAL - compareTo

Solution-1

Figure 15: Final fill-in-the-blanks question on the exam main page.

DO NOT RELOAD THIS PAGE!

If you accidentally do it, you have to contact your exam supervisor.

Submission:

Last Name + First Name

Exam: Creating a fill-in-the-blanks question

26.8.2022, 15:17 - 15:17

MatNr: PreviewSubmission

Fill-in-the-blanks question

This exam shows how to create a fill-in-the-blanks question.

20 Punkte | 20 Points

Comparable

Complete the Student class

```
public Student implements  {  
  
    String name;  
    int age;  
  
    public Student(String name, int age) {  
          
    }  
  
      
    public int compareTo(Student other) {  
        int comp = Integer.compare(age, other.age);  
        if(comp ==  ) {  
            comp = name.compareTo(other.name);  
        }  
        return comp;  
    }  
}
```

Figure 16: Exam with a fill-in-the-blanks question.

6 Related Work

One of the most popular online learning platforms at universities is Moodle [3]. Moodle (Modular Object-Oriented Dynamic Learning Environment) is a web-based and open-source learning content management system. It can be downloaded, used and modified for free. It is designed to help lecturers to create a personalized learning environment with high-quality lessons. With the help of Moodle students can learn independently of time and place [6]. It offers the lecturer several flexible tools such as sharing course materials, uploading videos, managing assignments or creating surveys. Interaction between lecturer and students as well as between students is also supported with the use of feedback and forums [5].

Moodle offers 15 different question types for creating exams. Among these question types, there is also a fill-in-the-blanks question. In Moodle, a fill-in-the-blanks question is called embedded answers (Cloze) and has the same blank types as our tool, just with different names: (1) short answer blanks (text blanks), (2) numerical blanks and (3) multiple choice blanks (dropdown blanks) [3]. Moodle uses a similar approach to our first approach using keywords (see in Section 3). The keywords `{` and `}` are used to mark the start and the end of the blank. A fill-in-the-blanks question in Moodle can look like this `{10:SHORTANSWER:=int} age = 24;`. This converts `int` to a text blank. Ten point score can be obtained for this question.

7 Current Limitations and Future Work

In this section we discuss the current limitations of our work and provide additional ideas for improving Xaminer.

Allowed number in numerical blank

Currently, only positive and negative numbers are allowed in the numerical blank. However, sometimes we might need other numbers such as power numbers (e.g. 10^2) or logarithms. Therefore it would be useful to extend the allowed numbers.

Keep preview next to blank editors

At the moment a question's preview is in a fixed position on the right-hand side of the screen. If we have to scroll down to a blank editor we may not see the preview anymore. This can quickly cause you to lose track. This may especially happen for questions with many blanks, as this leads to many blank editors. This feature requires the preview to slide along the top edge when scrolling.

Highlighting the currently edited blank

Another helpful feature that would make editing a solution in the blank editor easier is highlighting the blank in the preview when editing the solution in the blank editor. Thus, we know exactly which blank we are editing. To implement this feature, the *Keep preview next to blank editors* function must already be implemented.

Points for each blank

We can only give an overall point for the entire fill-in-the-blanks question. But it would be useful if we could assign points for each blank. To do this we have to add another text field to the blank editor where we can specify the points for the blank.

Tags

It would often be helpful for lecturers to search for a question in old exams quickly. One feature that could be implemented in the future to make this possible would be to assign tags to exam questions. It should be possible to assign multiple tags to each exam question, e.g. the fill-in-the-blanks question in Figure 15 gets the tag *Comparable* and the tag *2022*. In another window we can then filter exam questions by tags. All exam questions containing all specified tags will then be displayed.

8 Conclusion

Online exams will play an important role even after the coronavirus pandemic, especially at universities. There are many advantages of online exams for both students and instructors. For example students benefit from useful features such as syntax highlighting and lecturers from automatically corrected exams.

One tool that can be used to create, hold and correct online exams is Xaminer. It was developed by the Institute for System Software at the Johannes Kepler University Linz. With this thesis it is now possible to create fill-in-the-blanks questions in Xaminer. The fill-in-the-blanks questions can consist of different blank types, such as text blanks, numerical blanks or dropdown blanks. With a user-friendly graphical editor creating and editing the question is very comfortable and easy.

List of Figures

1	Overview of an example of a fill-in-the-blanks question	9
2	General part	10
3	Creating a fill-in-the-blanks question	11
4	Blank editor	12
5	Header	13
6	Header edit mode	13
7	Body of a text blank	14
8	Body of a numerical blank	15
9	Body of a dropdown blank	15
10	Visualizing a text blank	16
11	Visualizing the fill-in-the-blanks question type	18
12	Main page	19
13	Exam main page	20
14	Example of a fill-in-the-blanks question	22
15	Final fill-in-the-blanks question	23
16	Exam with a fill-in-the-blanks question	24

References

- [1] Bootstrap-vue documentation. <https://bootstrap-vue.org/docs>. Accessed: 2023-03-24.
- [2] Markdown guide. <https://www.markdownguide.org/>. Accessed: 2023-03-08.
- [3] Moodle documentation. <https://docs.moodle.org>. Accessed: 2023-03-08.
- [4] Vue documentation. <https://vuejs.org/guide/introduction.html>. Accessed: 2023-03-24.
- [5] K. Buntak, M. Mutavdžija, and M. Kovačić. Differences of e-learning systems with the focus on moodle and blackboard systems. *International Journal of E-Services and Mobile Applications (IJESMA)*, 13(1):15–30, 2021.
- [6] A. Zkear and B. Majeed. Smart learning based on moodle e-learning platform and development of digital skills for university students. *International Journal of Recent Contributions from Engineering, Science & IT (iJES)*, 10(1), 2022.