Interactive Programming Tutorials in Automated Programming Assessment System Edgar

I. Mekterović*, Lj. Brkić**, M. Fertalj*** and M. Fabijanić****
*, **, *** University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
**** Algebra University College/Software Engineering, Zagreb, Croatia
* igor.mekterovic@fer.hr
** ljiljana.brkic@fer.hr
*** melita.fertalj@fer.hr
**** mario.fabijanic@algebra.hr

Abstract - **Automated programming assessment tools are software systems widely used in education to assess programming code without manual intervention. Beyond exam scenarios, these tools are increasingly applied in e-learning contexts. In this realm, interactive programming tutorials have gained prominence for their effectiveness in teaching programming concepts. These tutorials blend theoretical knowledge with hands-on exercises, providing real-time feedback on code errors to facilitate prompt identification and correction by learners. The interactive nature engages learners actively, enhancing their understanding, and adaptability accommodates individualized progress. The accessibility and scalability of interactive programming tutorials suit learners of diverse skill levels. Integrated with automated assessment systems, interactive tutorials not only provide a dynamic and personalized learning experience but also alleviate the burden on instructors by enabling interactive content creation and offering valuable learning analytics. This paper introduces an evolution of the Edgar system, now equipped with an integrated interactive tutorial module. This module can evaluate embedded questions and code playgrounds in various programming languages, including SQL, Java, C, Python, etc., as well as multiple-choice questions. The integration represents a practical shift in programming education, offering learners a versatile and personalized approach to acquiring essential skills.**

Keywords - component; formatting; style; styling; insert (key words)

I. INTRODUCTION

Programming education has witnessed a significant shift towards technology-driven approaches, particularly with the widespread adoption of automated programming assessment tools. Automated programming assessment system (APAS) is an information system used in educational environments to (semi)automatically assess students' answers to programming questions. They typically also support other types of questions, such as multiple-choice questions, and provide monitoring and logging of exams, various statistics, and visualizations, etc. [1] Nowadays, they are typically implemented as web applications. An excellent overview of automatic grading and feedback tools can be found in a recent systematic review [2]. These tools excel in efficiently evaluating student code beyond exam settings, paving the way for their integration into e-learning environments. Within this landscape, interactive programming tutorials have emerged as powerful instruments for effectively conveying programming concepts (e.g. [3], [4], [5]). They can blend theoretical knowledge with hands-on practice, offering valuable near real-time feedback on code errors to enhance learning efficiency [2]. The interactive nature actively engages learners, fostering deeper understanding and adaptability to diverse learning styles. Also, the scalability and accessibility of these tutorials cater to learners across various skill levels. For the content creators, i.e. teachers, this technology also provides numerous advantages. Tutorials with integrated automated assessment features can automatically grade student code, freeing up teachers' time for other tasks like providing personalized feedback or creating new content. Consistent automated feedback ensures all students receive similar guidance, reducing the need for repetitive explanations. Interactive tutorials can cater to large numbers of students without requiring individual attention from the instructor, which is very important given the current negative trend of shortage of teaching staff in the IT sector. Furthermore, digital learning platforms can gather valuable data about students and their learning habits which can provide a foundation for various learning analytics systems, which is an area that is still under active development [6].

Recognizing this potential, in this paper we present an evolution of the automated programming assessment system Edgar with interactive learning capabilities. We introduce an integrated interactive tutorial module capable of evaluating various embedded elements within the tutorial page (step). This module can evaluate code in multiple programming languages (SQL, Java, C, Python) and supports diverse question formats: traditional multiple-choice questions, automatically evaluated programming questions and embedded code playgrounds. This integration presents a significant step forward for Edgar, offering learners a versatile and personalized approach to acquiring essential programming skills while at the same time equipping teachers with new digital platform for content creation and distribution – a very practical contribution to enhance programming education through technology-driven solutions.

II. EDGAR AUTOMATED PROGRAMMING ASSESSMENT SYSTEM

Edgar APAS [7] has been in development since 2016. at the Faculty of Electrical Engineering and Computing and is used for automatic evaluation of program code. It was originally developed for the needs of evaluating code in the SQL programming language (named after great Edgar F. Codd), but it has since evolved into a general-purpose system for evaluating program code. It can now evaluate code written in C, Java, Python, C++, etc., or in any programming language [1]. Besides programming questions, it supports eight other question types (multi-correct multiple-choice questions, free text questions, diagrams, etc.) and can even be used to conduct peer assessments [8]. Edgar is used intensively and has become an indispensable part of large courses with several hundred enrolled students (e.g., Introduction to Programming, Object-Oriented Programming, Databases), as well as smaller ones (e.g., Business Intelligence). For example, in the previous academic year 2022/2023, Edgar was used to conduct 74,513 exams with a total of 425,052 questions from 20 different courses. It is implemented as a web-application that relies on modern open-source technologies (Node.js, PostgreSQL, MongoDB, Angular, etc.) and is publicly available to everyone under the MIT license [9]. In order to extend Edgar with interactive tutorials we had to develop a new SPA (Single Page Application) application, the corresponding API and, of course, modify the database. Figure 1. shows high-level modular architecture of Edgar APAS. The newly developed SPA is shown in the top left corner. Of course, this extension also required a new server-side API and database modifications.

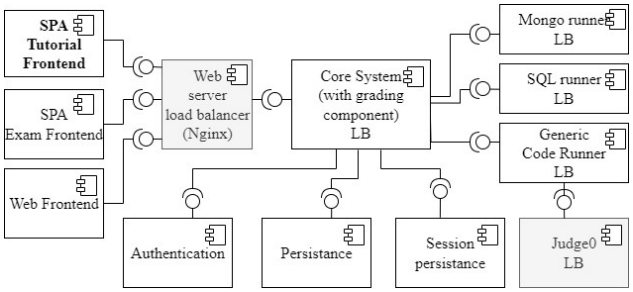

Figure 1. Edgar's modular architecture extended with Tutorial SPA (upper left corner), new API and database modifications. 3rd party components show in gray.

III. INTERACTIVE TUTORIALS IN EDGAR

Given that Edgar has rich possibilities for performing and evaluating program code, it is natural to develop a system in such a way that, in addition to testing students, these possibilities are also used for learning. Such an approach is advantageous for teachers as well (an alternative would be to use commercial 3rd party cloud services) – all their digital materials are in one place and under their control. Students also have a better user experience because they access one platform with a uniform interface. A tutorials module was developed that allows teachers to define tutorials (digital lessons) simply and easily. Students, on the other hand, have a simple and functional learning application: Figure 2. shows the tutorial layout with simple header with basic information about the step and navigation controls (back, forward and direct step selection via drop-down). The content is laid out vertically. Also, note the ticket button which is used to communicate with the content-creator. This is a very important feature as it enables us to collect feedback from the learners, helping identify potential errors or topics that are demanding and need to be explained more clearly.

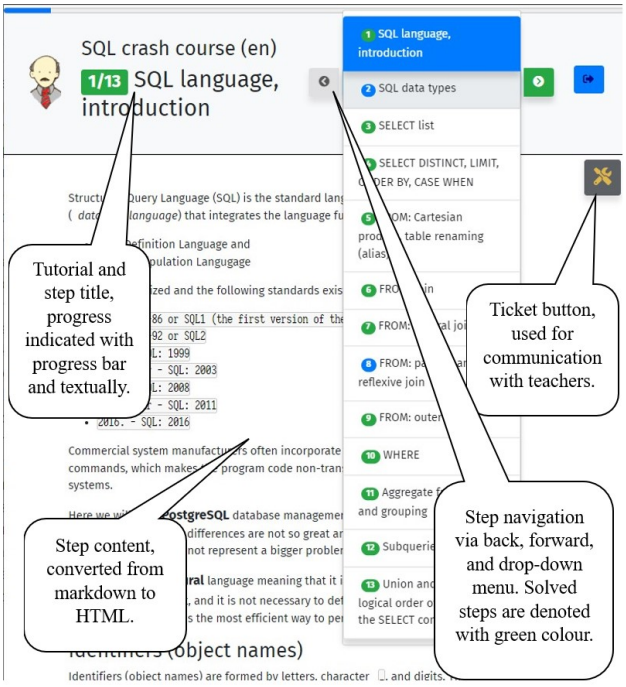

Figure 2. Tutorial layout: simple and clear header with tutorial and step information, and navigation controls. Step content is laid out vertically. Ticketing system is included for communication with the teacher.

Obviously, tutorials consist of an arbitrary number of steps. Step is the basic building block of the tutorial, and its content is defined using the markdown language. Markdown is a simple markup language used to format text documents with plain text characters. It allows one to easily define titles, bold text, italics, lists, etc. without the need for special software or programming knowledge. Markdown is simple, searchable, and readable in its native form, which makes it popular for writing online content, creating readme files, and managing notes. There are several versions of markdown, and Edgar already uses GFM (GitHub Flavored Markdown) [10] to define the question text, which is then transformed into HTML and displayed to students in the browser. Additionally, in Edgar GFM is extended to support constructs like collapsible elements, mathematical formulas in Latex syntax and various diagrams (Sequence, Class, ER, Flowchart, etc.). For instance, Figure 3. shows the definition of a question in Edgar where both formulas (Latex expressions are enclosed in $$) and flowchart diagram are used, followed by syntax highlighted programming code in C.

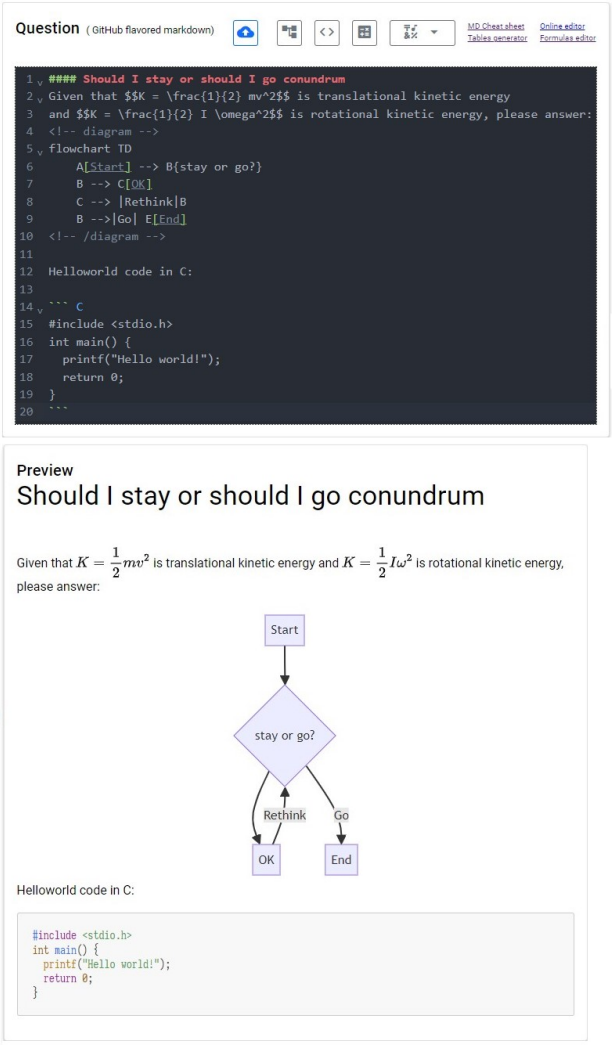

Figure 3. Extended GFM markdown in Edgar – markdown is at the top of the figure, and rendered HTML preview at the bottom of the figure

Building on that, and to support the embedding of questions in the content of the tutorial steps, new extensions to GFM need to be introduced. Since GFM is translated into HTML and it is allowed (but not recommended) to embed HTML expressions in GFM, HTML comments were used to define these new constructs. Comments are ignored in the initial markdown processing step and can be postprocessed by Edgar's code to produce desired content. In the remainder of the chapter, different types of embedded questions are described.

A. Multicorrect multichoice questions

There are two ways to embed a multiple-choice question into the tutorial step content:

(1) **Embedded multicorrect multichoice questions** – these are regular questions that are inserted into the step text via the following markup:

```
<!-- question qid=<ID> {required} -->
<!-- /question -->
```

Obviously, an existing question (ID) from Edgar's question bank must be used, so this approach requires an extra step of creating a question in Edgar's regular question edit form.

Any question can have the "required" attribute set, which indicates that the question must be answered if the tutorial is such that it does not allow skipping steps. With this, teachers can force students to answer mandatory questions to complete the entire tutorial.

(2) **Inlined (markup) multicorrect multichoice questions** – for multicorrect questions, creating a new question might be an overkill, especially for simple questions that will not be used elsewhere in exams, and so it is possible to create an ad-hoc question via markup:

```
<!-- multichoice required -->
<Question text>
@@@
Option 1
@@@
Option 2 <!-- correct -->
@@@
Option 3
<!-- /multichoice -->
```

The number of options and number of correct options is arbitrary.

The following code defines a step with two multichoice questions and Figure 4. shows how they are rendered to the screen.

```
#### (a) Multichoice question 48135 referenced from
Edgar's question bank:
<!-- question qid=48135 required -->
    // inital code here, delete "required" attribute above
if the question is optional
<!-- /question -->

#### (b) Embedded multicorrect multichoice question:
<!-- multichoice required -->
Should I stay or should **I go**?
@@@
Stay <!-- correct -->
@@@
Go <!-- correct -->
@@@
Stall
<!-- /multichoice -->
```

Note the bookmark dots on the right serving two purposes: they change color depending on whether the question is answered correctly and enable quick positioning (focusing) on the question.

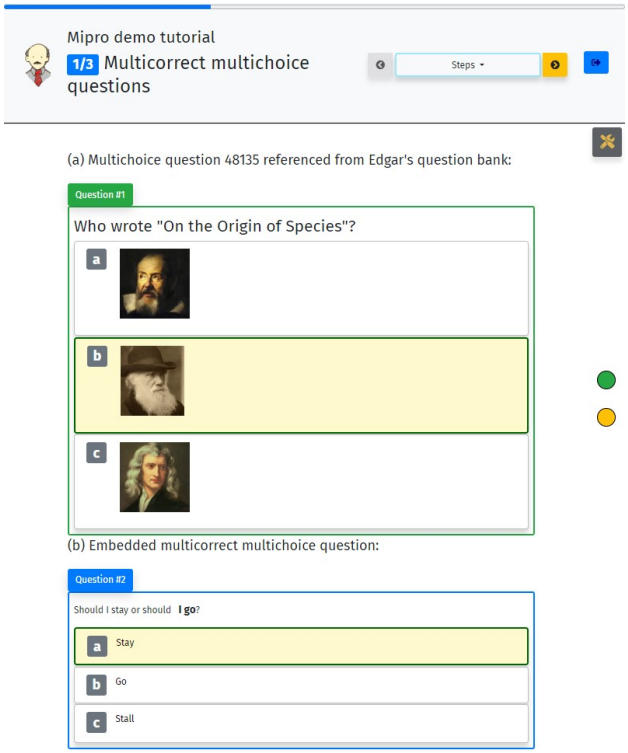

Figure 4. Embedded and inlined multicorrect question. Bookmark dots on the right serve two purposes: they change color depending on whether the question is answered correctly and enable quick positioning (focusing) on the question.

B. *Code questions*

Given that code questions are more demanding in terms of data that needs to be defined (answer in a programming language, N test-cases, etc.) [7], it does not make sense to enable an "ad-hoc" inline version as in multi-correct questions, and such questions must be defined in a standard way and then simply referenced from the step markdown. This approach also allows us to use all previously defined questions. The extended markdown syntax is as follows:

```
<!-- code-question qid=<ID> {required} -->
// initial code here
<!-- /code-question -->
```

It is possible (not mandatory) to specify an arbitrary text between the code-question tags that will then appear as the initial program code in the tutorial. SQL questions are referenced in the same way as questions from "classic" programming languages (C, Java, ...), although the way they are rendered and evaluated is fundamentally different [7]. For example, Figure 5. shows a C question and a SQL question for the following markdown:

```
### 2. **Code question** - you can use any programming
language supported in Edgar, eg. C, Java, Python, SQL, ...

<!-- code-question qid=41557 -->
// inital code here
double pi(int n) {
}
<!-- /code-question -->

Note that this question does not have the "required" flag.
So, the entire step will turn green with or without the
correct solution to this question.
Also, if this was allow-random-steps-OFF tutorial, the
solution to this question would be optional.

### SQL questions
SQL is also supported, eg:
<!-- code-question qid=46610 required -->
<!-- /code-question -->
```

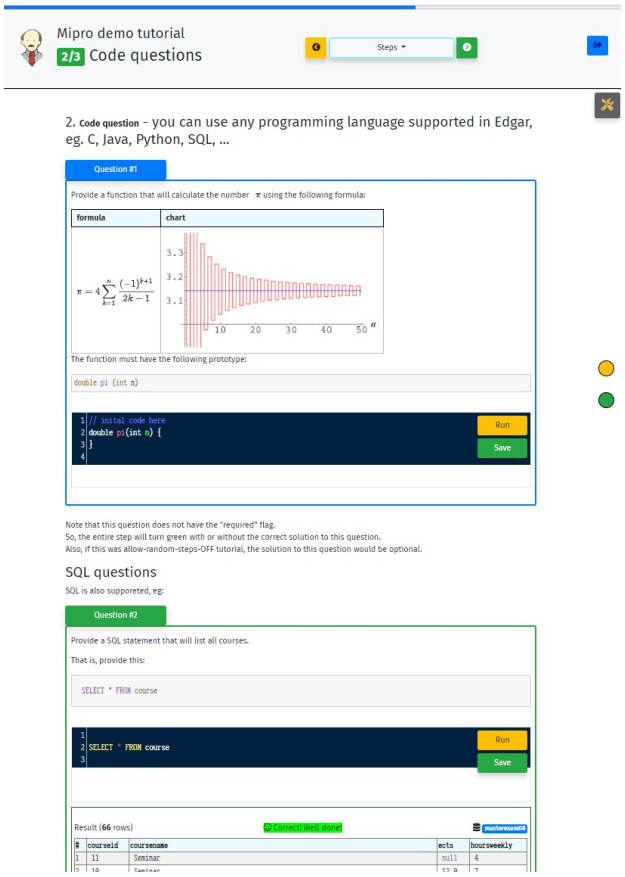

Figure 5. C and SQL questions. C question in optional, while SQL question (solution) is required.

Note that the first (C) question does not have the "required" flag - the entire step will turn green with or without the correct solution to this question. Also, if the tutorial is defined to require step solutions to traverse steps, this question will not be considered. In this way, we can have N mandatory and M optional questions in any step.

C. *Code playgrounds*

Finally, sometimes it is convenient to just allow the student to try some code, whether it is already prepared or needs to be written or expanded on the given template. Then it is convenient to allow the student to have a "small development interface" within the step, where it is possible to write code, provide input and run the code and see the results. This is very similar in functionality to the commercial online Read-Eval-Print Loop systems (e.g. ReplIt [11], CodePen [12], etc.) that allow users to try out the code online, only here it is elegantly integrated into the tutorial content. The syntax is slightly different than the question syntax:

```
<!-- code-playground crid=<ID> langids=(<ID>{, ID}*) -->
{initial code}
<!-- /code-playground -->
```

where crid stands for code-runner ID and is tied to the way Edgar evaluates code – it sends it to the services which "know" how to evaluate some programming languages and these services are registered in Edgar with their IDs. Langids stands for "language IDs" and denotes a list of allowed programming languages that can be used to write the code. Initial code is optional. Edgar's tutorial definition GUI has buttons that insert these markups into the markdown, so the user does not really need to know them in detail. Figure 6. shows code playgrounds for the following markup:

```
<!-- code-playground crid=10 langids=(4)-->
// Please fix the following code
// there is a syntactical and logical(security) error in this code:
#include <stdio.h>
int main()
{
    char[512] stdin;
    printf("\nHello from the playground!");
    scanf("%s", stdin);
    printf("\nYour stdin was: %s", stdin);
    return 0;
}
<!-- /code-playground -->
---
Code questions and playgrounds are also available in multi-language
forms!
Try this hello world example for Python and C:
<!-- code-playground crid=10 langids=(4,34) -->
   write hello world
<!-- /code-playground -->
```

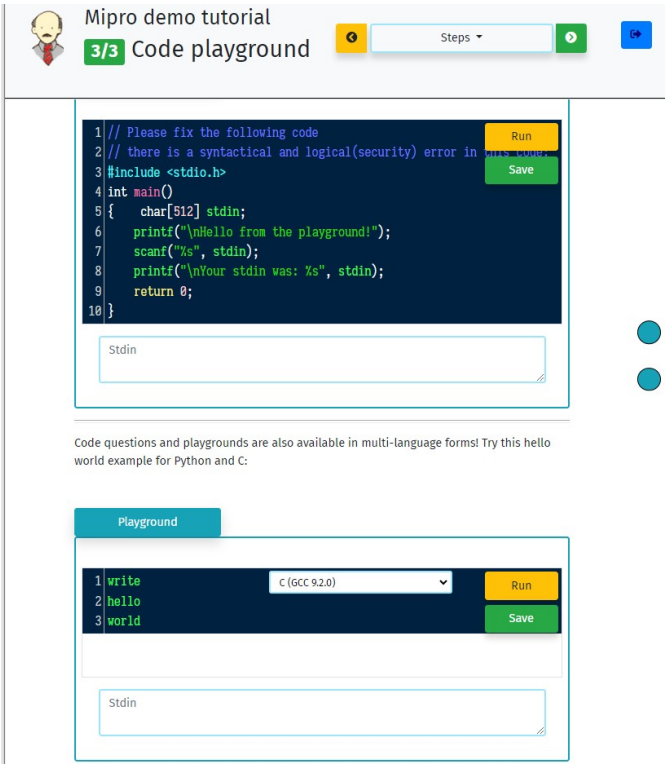

Figure 6. Single programming language and multi-programming language (C and Python) sandbox.

D. *Learning analytics foundation*

Besides keeping the tutorial state for each student (current answers, resolved steps, etc.) so that students can leave and continue tutorials later, Edgar keeps a very detailed log of activities in the tutorial application which can provide a foundation for learning analytics scenarios. Figure 7. shows a part of MongoDB document that is kept for each student-tutorial instance – note the "events" array which stores the student activity.

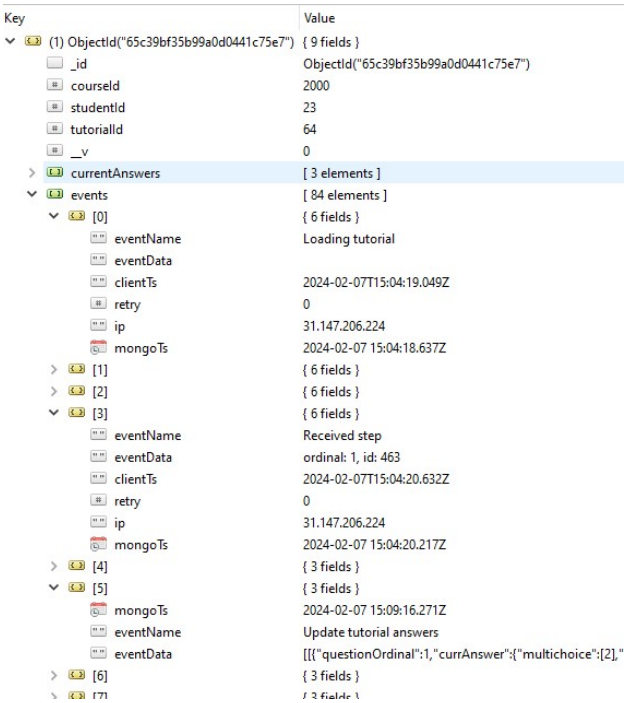

Figure 7. Student's tutorial document – besides current answers, it contains the detailed log of activites (events array)

This valuable data quickly accumulates, especially for large courses. Figure 8. shows the tutorial usage for the Databases course in the past academic year. We can see the trend of decreased usage as the semester progresses.

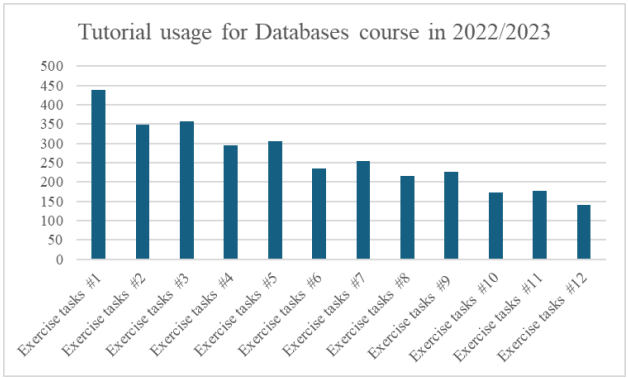

Figure 8. Usage for the Databases course in 2022/2023 showing negative trend in usage – 439 students used first tutorial, and only 141 last tutorial.

Cumulatively, that makes 3168 documents like the one in Figure 7. that can be used to provide various analysis – from preventing student churn, predicting the outcome of

the course, improving the tutorial content, etc. especially when combined with other data collected by the APAS. These topics will be the subject of a future work.

IV. CONLUSION

This paper introduces an innovative enhancement to the Edgar automated programming assessment system: an integrated interactive tutorial module. This module empowers educators to create engaging learning experiences by incorporating diverse question formats and code playgrounds within the tutorials. It leverages existing APAS functionalities like secure code execution and ticketing system to support various programming languages, including SQL, Java, C, and Python, catering to a broader learner audience. The key benefits of this integration are: (1) versatility and personalization: students can progress through interactive tutorials at their own pace, encountering various question types and practicing code in real-time, promoting individualized learning; (2) enhanced engagement: the interactive nature of the tutorials actively engages learners, fostering deeper understanding and adaptability to diverse learning styles; (3) content creation and distribution: teachers benefit from a user-friendly platform for creating and sharing interactive tutorials, streamlining content creation and distribution, and (4) learning analytics foundation: Edgar's detailed activity logs provide valuable data for future learning analytics initiatives.
In a more general sense, our work demonstrates how existing APAS systems can be extended to seamlessly integrate interactive teaching lessons. This leverages the inherent strengths of APAS, such as sandboxing and code evaluation, to create engaging and personalized learning experiences. Moreover, the rich data already collected through assessments and now within the interactive elements opens exciting avenues for future research in learning analytics. This paves the way for personalized learning scenarios and improved educational outcomes. In addition, in future work, we will explore the possibilities of integrating Edgar with generative AI models to provide students with fast and high-quality feedback. We will investigate the usability of AI models for assessment as well, but keeping in mind that this technology is readily available to everyone, it is also necessary to address plagiarism detection.

REFERENCES

[1] I. Mekterovic, L. Brkic, and V. Krstic, "Programmable Questions in Edgar," *2023 46th ICT Electron. Conv. MIPRO 2023 - Proc.*, vol. 0009, pp. 775–780, 2023, doi: 10.23919/MIPRO57284.2023.10159897.

[2] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated Grading and Feedback Tools for Programming Education: A Systematic Review," *ACM Trans. Comput. Educ.*, vol. 1, no. 1, pp. 1–43, 2023, doi: 10.1145/3636515.

[3] R. Suzuki, J. Kato, and K. Yatani, "ClassCode: An Interactive Teaching and Learning Environment for Programming Education in Classrooms," 2020, [Online]. Available: http://arxiv.org/abs/2001.08194.

[4] F. Engelberger, P. Galaz-Davison, G. Bravo, M. Rivera, and C. A. Ramírez-Sarmiento, "Developing and Implementing Cloud-Based Tutorials That Combine Bioinformatics Software, Interactive Coding, and Visualization Exercises for Distance Learning on Structural Bioinformatics," *J. Chem. Educ.*, vol. 98, no. 5, pp. 1801–1807, 2021, doi: 10.1021/acs.jchemed.1c00022.

[5] Z. Azimullah, Y. S. An, and P. Denny, "Evaluating an interactive tool for teaching design patterns," *ACE 2020 - Proc. 22nd Australas. Comput. Educ. Conf. Held conjunction with Australas. Comput. Sci. Week*, pp. 167–176, 2020, doi: 10.1145/3373165.3373184.

[6] L. Márquez, V. Henríquez, H. Chevreux, E. Scheihing, and J. Guerra, "Adoption of learning analytics in higher education institutions: A systematic literature review," *Br. J. Educ. Technol.*, no. August, pp. 1–21, 2023, doi: 10.1111/bjet.13385.

[7] I. Mekterovic, L. Brkic, B. Milasinovic, and M. Baranovic, "Building a comprehensive automated programming assessment system," *IEEE Access*, vol. 8, pp. 81154–81172, 2020, doi: 10.1109/ACCESS.2020.2990980.

[8] L. Brkić, I. Mekterović, M. Fertalj, and D. Mekterović, "Peer assessment methodology of open-ended assignments: Insights from a two-year case study within a university course using novel open source system," *Comput. Educ.*, p. 105001, Feb. 2024, doi: 10.1016/J.COMPEDU.2024.105001.

[9] L. Mekterovic, Igor; Brkic, "Files · master · edgar-group / edgar · GitLab." https://gitlab.com/edgar-group/edgar/-/tree/master (accessed Feb. 06, 2024).

[10] "GitHub Flavored Markdown Spec." https://github.github.com/gfm/ (accessed Feb. 07, 2024).

[11] "Home - Replit." https://replit.com/~ (accessed Feb. 08, 2024).

[12] "CodePen: Online Code Editor and Front End Web Developer Community." https://codepen.io/ (accessed Feb. 08, 2024).