Online testing platform with eye tracking

Milan Segedinac, Goran Savić, Igor Majić
Faculty of Technical Sciences, Novi Sad, Serbia
milansegedinac@uns.ac.rs, Savicg@uns.ac.rs, majic753@gmail.com

Abstract— **In this paper we present a platform for online testing which tracks students' gazes. *EyeTrackQuiz* enables a teacher to develop quizzes where every individual question can be fine-tuned via the editor which supports freely positioning of different elements (titles, introduction paragraphs, images, possible answers and code fragments). Furthermore, the editor supports defining regions of interests (ROIs). These regions represent rectangle regions of the screen, which the teacher considers important. While the student is taking the test, his eye movements is tracked by an eye tracker which is connected to the computer. Only gazes which fall into the regions of interest are included in the final reports. After the test conclusion, it is possible to see the results and eye tracking reports. Such reports give a better insight into student's cognitive processes. The solution is implemented as an in-browser application, with an integrated software library for communication with the eye tracker.**

Keywords—*eye tracking, online testing, educational platform, web applications*

I. INTRODUCTION

In the educational system, the development of high-quality standardized tests is a complex process. If the questions require the student to supply answers in form of short essays, the grading is left to the teacher, as the domain expert. In this case the grading process is highly subjective and quite demanding for both the student and the teacher [1]. If the grading process is based on multiple choice questions, it can be standardized, reducing the teacher's involvement to a minimum. However, the downsides of this process stem from the fact that it is difficult to determine the number of lucky guesses, as well as the percentage of accidental errors. If the information extracted from an *eye tracking* device were to be incorporated into the grading process, the effect of these downsides would be reduced significantly.

One sided way of teaching dominates education. It means that the teacher is assigned a dominant role, and as a result the teaching sessions are not interactive. Personal computers (PCs) have become a ubiquitous part of modern life. Despite this, they remain underutilized in education [1]. Despite requiring a significant amount of investment, new technology contributes to the efficiency of modern education. Furthermore, the technology allows the teacher to dedicate himself to the more creative aspects of his job, while at the same the students' performance improves significantly [1] [2].

Testing represents an evaluation of student's knowledge. It is a pillar of the education process and serves to evaluate and control it. Standardized testing is a complex problem, for which there is no general-purpose solution. In many fields of study, the teacher, as the domain expert, is in charge of the whole grading process. The process is error-prone with a plethora of error causes such as lack of concentration, tiredness and grader bias.

Standardized testing, in form of questions with multiple choice answers, eliminates aforementioned problems and uses modern technology to automate the grading process. The main drawback of this approach is the inability to determine the exact number of answers which were picked randomly, as well as the student's reading comprehension of the questions and answers presented. In order to provide enough insight into students' cognitive processes, these pieces of information are required: amount of time a given student has spent on a question, parts of the question which caught his attention (question title, additional text, images and individual answers), and the exact order of elements subject to his attention.

In order to be able to monitor cognitive processes during evaluation, in this paper we propose a testing platform with eye tracking. Aside from enabling test taking and numeric results recording, it also features student's eye movement recording during evaluation, thus providing insight into his cognitive processes.

The paper is divided into five sections. In Chapter II related work in comparison with the provided solution is discussed. Chapter III presents the software architecture of the proposed platform. Chapter IV discusses the implementation details, while Chapter V is a conclusion summing the contents of the paper. Furthermore, that chapter contains several suggestions for broadening the research and improving the discussed platform.

II. RELATED WORK

Eye tracking is an act of tracking eye position on the screen. It is based on optical tracking of corneal reflexes. The concept itself is relatively simple, unlike the technology employed to realize it [3].

An *eye tracker* is a specialized hardware used for eye tracking. It utilizes infrared cameras to track the user's gaze. It is connected to a laptop or desktop PC via USB [4].

Various forms of eye tracking have existed for decades. Recently, rapid breakthroughs were achieved using more sophisticated technology. This technology has a multitude of applications, including but not limited to education [4].

In [5] the authors present an eye tracking plugin for the integrated development environment called *Eclipse*. It is an extension of the *iTrace* plugin. It is not limited to evaluating shorter segments, making it applicable in larger, enterprise projects. It utilizes a dedicated Java library to communicate with the eye tracker. Furthermore, it can distinguish between code and non-code elements. Finally, when exporting data into JSON format, it can filter out noise. Testing sessions, featuring volunteers showed promising results. The main difference in relation to the platform

described in this paper is that it is a highly specialized plugin, interpreting gazes on programming code. By contrast, this paper describes an education platform, implemented as a standalone web application, which utilizes an eye tracking device in the evaluation process. Similarly, our web application also features a dedicated library for communication with the eye tracker. Moreover, it also allows for gaze data to be exported into easily readable formats.

Paper [6] is a study done on two subjects studying a short code snippet and answering different questions about it. Their gazes were recorded using an eye tracking device and exported as video files (*.avi*). These video files were then analyzed in a workshop. The purpose of the workshop was to determine whether eye tracking is feasible, can provide enough data and lead to application of new ideas in area of education. The researchers developed the concept of code schemes, featuring elements such as line of code, code block, method signature and method call. This concept maps to the regions of interest (ROIs) in our web application. Unlike the scheme, ROIs are completely user defined, both in screen area they cover, and their name. The workshop's conclusion was that different experts can have radically different gaze patterns, even when reading the same code. On the contrary, our application doesn't perform any pattern recognizing. However, its export feature could be used to provide input into a model, which could determine a pattern where humans fail to see one.

In paper [7] *eye tracking* was applied to measure students' attention spans when solving multiple choice science questions. The domain of the problem was landslide prediction with four possible answers and four factors (factors are images representing temperature, rainfall, slope and debris). Both this solution and our web application provide a way to integrate eye tracking into the evaluation process and both feature tests with multiple choice answers. However, there are no domain constraints present in our web application, so it allows for testing an arbitrary number of hypotheses. After applying several statistical methods, the researchers came to the following conclusions. The examinees, in general, tend to pay more attention to chosen options than the rejected alternatives. Furthermore, they spend more times examining relevant factors. The main difference between this study and the platform presented in this paper is its great reliance on statistical methods and a requirement to think aloud, which was placed on the examinees. In contrast, the web application doesn't have this requirement as it doesn't record audio.

III. System Architecture

This section presents the software architecture of the proposed assessment platform with eye tracking (*EyeTrackQuiz*), while the more detailed description of the platform's functionalities will be given in the implementation section. An overview of application components is present in *Fig 1.* Each of them is explained in the following paragraphs.

Eye tracker [4] software is used for calibrating the device before use. The device used is *Gazepoint GP3 eye tracker.* To perform the calibration successfully it is necessary to register at least four out of five positions (performed by looking at the points with both eyes) shown on the screen by the device's software. Afterwards the calibration settings are persisted and the device can be used. Furthermore, the software also acts as a server, and is able to send data in form of XML fragments.

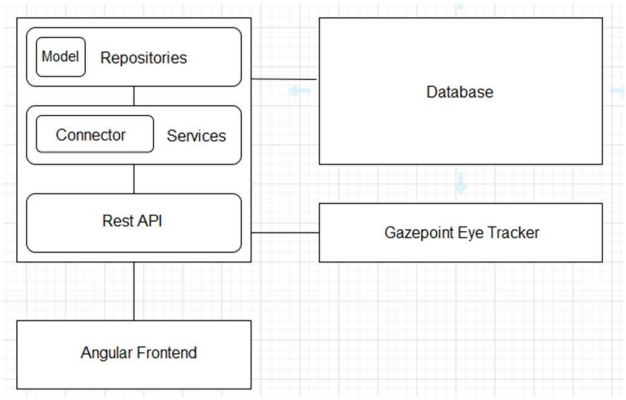


Figure 1: Application schema

The *Connector* library was written in the Java programming language. Its purpose is to connect to the Eye tracker server software and forward data to the application. It is possible to configure eye tracker server IP address/hostname and port. After initial setup, where the application specifies which kind of data it would like to receive from the server, data streaming begins. The library handles conversion from XML fragments to Java objects and vice versa. These objects can then be utilized by the application. To facilitate this two-way conversion JAXB [8] parser is used. Implementation wise, aside from Java classes written to encapsulate the parsed objects, the library contains a main class (*GazepointConnector*), which handles communication, using TCP sockets from the standard Java library.

Backend application utilizes the *Connector* library to communicate with the *eye tracker* server. Aside from managing the data which arrives from the server, the backend application manages user data. It has a multilayer architecture, using the Spring Boot [9] framework.

REST layer receives the input data and sends the output data in form of JSON objects. It is implemented with classes using a declarative annotation mechanism from Spring Boot in an effort to reduce boiler plate code. This layer delegates all operations to corresponding services.

The service layer contains business logic. It performs all necessary validations of the data, prior to its persistence. In case of error the service layer throws exceptions. These exceptions propagate to the REST layer, where they are mapped to appropriate responses. The service layer relies on the repository to persistent and get data.

The repository layer stores and gets data and performs communication with the database. Hibernate object-relational mapper is utilized to streamline create, read update and delete operations (CRUD) on the data. The mapper also abstracts away the communication with the database by implementing Java persistence API (JPA). If an error or an exception occurs in this layer, it propagates

to the service layer, which handles it, and notifies the REST layer.

Frontend application communicates with the user and delegates his requests to the *backend* application. It is implemented as a single page application (SPA), meaning that, after the initial load user interaction is a fast and fluid experience. It is implemented using the *Angular* [10] framework. In order to achieve a more modern outlook, the *Angular Material* [11] library, which also provides the essential drag & drop functionality. This functionality makes direct manipulation of graphic elements possible. More detailed description is presented in the *Implementation* section.

While taking the test, the platform tracks student gazes. The tests are designed by teacher through the built-in test editor (see *Fig. 2*).

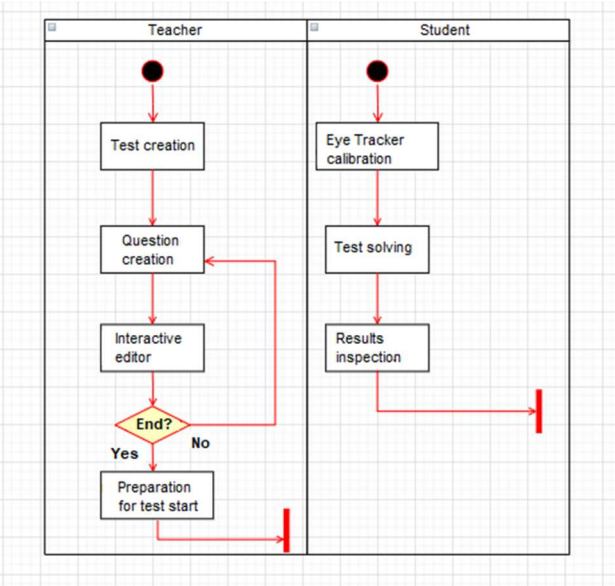


Figure 2: Application process diagram

The teacher inputs test name and formulates questions. Editing of each individual question is handled by an editor which allows placing and direct manipulation of various elements (Question title, introduction paragraph, answers, images and others). The editor also features defining and resizing of regions of interest (ROIs), which represent rectangular regions of the screen, which the teacher deems to be of significance.

The student takes a test (see *Fig. 2*). Since the app is connected with an eye tracking device, the acquired data is used for generating reports. These reports provide a deeper insight into student's knowledge and thought processes. The reports feature information whether the supplied answers were correct or incorrect, as well as relevant regions of interest data. This data can be downloaded as an *excel* file, which makes them suitable for further analysis and research.

IV. Implementation and Discussion

Application is organized into several sections which communicate and are dependent on one another. By looking at *Fig. 1*, we see that the *frontend* application communicates with the *backend* application by exchanging JSON objects. Moreover, the *backend* application utilizes the database to fetch and permanently store data. Furthermore, the application communicates with the Eye Tracker and processes the data using a dedicated library (*Connector*).

From a user's perspective the application is divided into two main sections. The first section enables creating and updating quizzes and their individual elements, while the second section enables testing, reports viewing and report exporting to *Microsoft Excel* format for further use.

In *Fig 3*, it is possible to see currently active quizzes. It is possible to create, update and delete elements in the table, by clicking on the corresponding actions. By clicking on the *edit questions* option, it is possible to do the same operations on the chosen questions (see *Fig. 4*).

Figure 3: Active quizzes

Clicking on the *edit question elements* opens an editor which allows for direct manipulation on individual question elements (see *Fig. 4*).

Figure 4: Quiz question list

In *Fig. 5*, it is possible to see a few types of elements. All types share the following common characteristics. The first characteristic is that their position can be updated with *drag & drop* manipulation. Furthermore, all elements are selectable / unselectable. A red frame serves as an indicator of currently selected element.

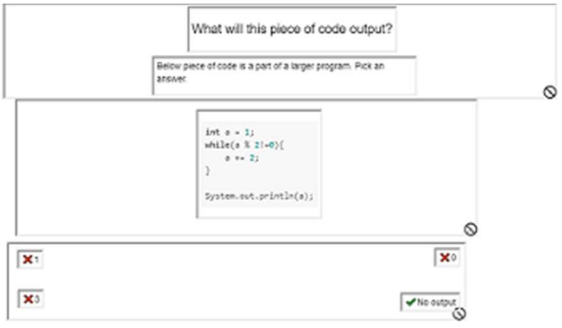


Figure 5: Interactive question editor

In order to delete an element, the user has to select it. Afterwards, it is possible to remove it by clicking *Remove selected element* button, or pressing the *Delete* key on the keyboard. Pressing the corresponding add button will open a dialog, where the user can input information about the element. This dialog also opens when double clicking on the element which allows the user to update it (see *Fig. 6*).

Different types of elements are supported. These elements are divided into two groups: fixed size and variable size.

Question title is a fixed size element. Its size depends on title length. Its most common use is to set up the question.

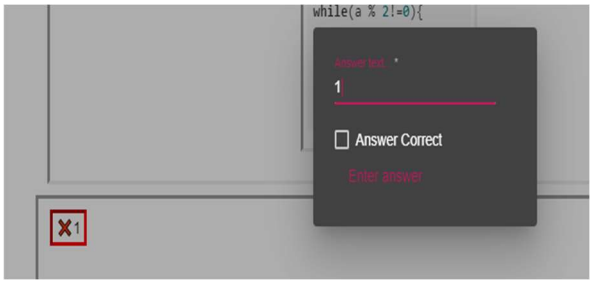


Figure 6: Element edit dialog

Question description is a fixed size element which provides additional explanation or context. It is implemented as a paragraph element.

Multiple choice answer is a fixed size element (see Fig. 6). It represents one of possible answers to the question and can be correct or incorrect. This setup provides flexibility, when making questions with zero, one or multiple correct answers.

Code element is a fixed size element which supports displaying short code snippets. This provides better support for programming domain quizzes.

Image element is a fixed size element which enables uploading of arbitrary images, further widening the quiz domain.

Second group of elements, with variable size consists of regions of interests (ROIs). Unlike other elements, they are not visible during testing, but student's gazes map on them. They contain a name (usually specific to test domain), position and size. Their shape is rectangular (see *Fig. 7*).

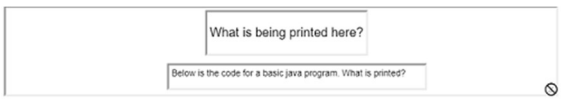


Figure 7: Region of interest example

As presented in *Fig. 7,* the region of interest groups question title and description. Clicking on the bottom right corner will freeze the region's position, allowing it to be resized. Regions of interest can overlap, although it recommended to keep them separated to make generated reports concise and easier to follow. Hovering over the region with the mouse cursor will reveal its name in a pop up.

Second section of the application is accessed by clicking on the *Play quiz* button. A form will pop up asking the user to input first name and last name which will be present in the generated final report. To include region of interest information, it is necessary to start and calibrate the *eye tracker* beforehand.

Questions follow their order of creation. All elements maintain their positions, which were set in the interactive editor (*see Fig. 8*).

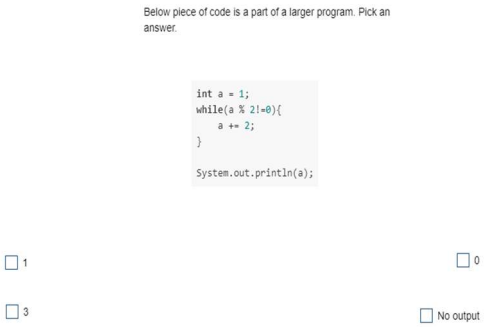


Figure 8: Question in test mode

Furthermore, regions of interest are not visible, while multiple choice answers are rendered as selectable checkboxes. It is possible to navigate through questions and answer them in arbitrary order using *Next* and *Previous* buttons (see *Fig. 9*). It is also possible to skip questions entirely. When navigating to the last question, instead of the next button the application will render a *Submit* button, which will end the testing session and show reports.

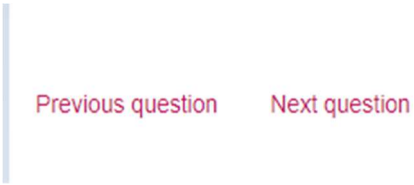


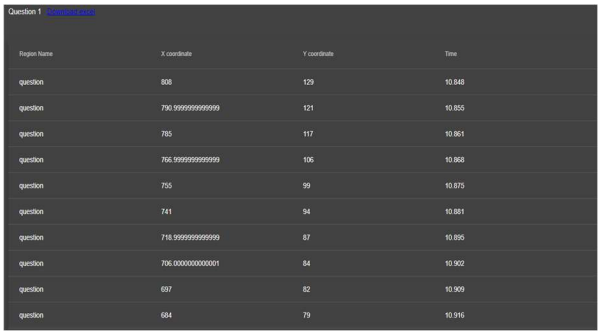
Figure 9: Question navigation panel

Report page consists of three tabs (see *Fig. 10*). All tabs are divided into collapsible subsections. Every subsection corresponds to an individual question.

Figure 10: Report tabs

First tab contains information about the student's chosen answers. The header of every subsection contains question name and the number of correct answers in relation to total number of correct answers. Clicking on the subsection will list all chosen answers with the correctness label.

Second tab contains raw data about the regions of interest. The header of every subsection contains question name and the download link for the data in Microsoft Excel (*.xlsx*) format. These data are suitable for further analysis and research. Clicking on a subsection shows a table with the following information: relevant region name, x and y gaze coordinates and gaze time (in seconds) from starting the eye tracker (see *Fig. 11*).



Question 1 download excel

Region Name	X coordinate	Y coordinate	Time
question	808	129	10.848
question	790.9999999999999	121	10.855
question	785	117	10.861
question	766.9999999999999	106	10.868
question	755	99	10.875
question	741	94	10.881
question	718.9999999999999	87	10.895
question	706.0000000000001	84	10.902
question	697	82	10.909
question	684	79	10.916

Figure 11: Standard ROI report

The third tab resembles the second. The data are also shown in tables and export to Excel is available. The main difference is that aside from regions of interest' names, there are two columns denoting start and end of the interval during which the student focused on a specific region. These data give a better insight into student's activity, allowing for deeper analysis of his knowledge.

V. CONCLUSION

In this paper a platform for creating tests and automatic grading of students was presented. Using the *eye tracking* technology, *EyeTrackQuiz* platform provides a more complete insight into student's cognitive processes. Furthermore, it provides a way to export data into a format suitable for further analysis and research.

The solution presented in this paper is robust. However, the research could be expanded in the following ways: support for *freeform* (not limited to rectangle or any other shape) regions, aggressive noise filtering in reports and resizing support for all elements in the interactive editor.

Furthermore, the system has the following limitations, which derive from the used eye tracking equipment. The eye tracker limits assessments to strict, laboratory conditions. Moreover, it forces the student to place his head in non-standard way, which can negatively impact his behavior and answers.

REFERENCES

[1] IBRAHIMOVIĆ, Samer. Obrazovna tehnologija i savremena nastava. Fakultet tehničkih nauka, Čačak, available on: http://journal. ftn. kg. ac. rs/download/SIR/SIR% 20Samer% 20Ibrahimovic% 20851_2012. pdf, cited, 2012, 28: 2016.

[2] MUSTAFIC, Mirza, 2014, Obrazovna tehnologija, http://www.ftn.kg.ac.rs/download/SIR/SIR%20Mirza%20Mustafic%208302014.pdf [accessed 09.05.2021]

[3] iMotions, 2019, *Eye Tracking,* https://imotions.com/blog/eye-tracking/ [accessed on 09.05.2021.]

[4] Gazepoint, 2010,*Gazepoint Eye Tracker*, https://www.gazept.com/ [accessed on 09.05.2021.]

[5] Mitrović, A., Vidović, M., Radosavljević, I., Mladenović, M., Savić, G., Segedinac, M., Konjović, Z. *Software for an eye tracking device enabling analysis of a student's interaction with program code*. In: Konjović, Z., Zdravković, M., Trajanović, M. (Eds.) ICIST 2018 Proceedings Vol.1, pp.128-132, 2018

[6] BUSJAHN, Teresa, et al. Eye tracking in computing education. In: Proceedings of the tenth annual conference on International computing education research. ACM, 2014. p. 3-10.

[7] TSAI, Meng-Jung, et al. Visual attention for solving multiple-choice science problem: An eye-tracking analysis. *Computers & Education*, 2012, 58.1: 375-385.

[8] Oracle, 2014, *JAXB java parser*, https://docs.oracle.com/javase/8/docs/technotes/guides/xml/jaxb/index.html [accessed on 09.05.2021.]

[9] Pivotal, 2014, *Spring Boot,* https://spring.io/projects/spring-boot [accessed on 09.05.2021.]

[10] Google, 2016, *Angular Framework*, https://angular.io/docs [accessed on 09.05.2021.]

[11] Google, 2016, *Angular Material*, https://material.angular.io/guides [accessed on 09.05.2021.]