



A Framework for the Generation and Execution of Multiple-Choice Questionnaires

DÁNIEL KOVÁCS¹

University of Miskolc, Hungary
Institute of Information Technology
kovacs435@gmail.com

ANITA AGÁRDI

University of Miskolc, Hungary
Institute of Information Technology
anita.agardi@uni-miskolc.hu

Abstract. The aim of the paper is to develop a website that provides the opportunity to create and complete multiple-choice quizzes, primarily to objectively measure users' knowledge. Although there are many similar platforms available on the Internet, they often have security flaws that make the system easy to circumvent, thereby undermining its meaning and reliability. The paper highlights the most common problems of the existing systems: the possibility of copying questions and answers, the possibility of modifying data stored on the website's frontend (e.g. correct answers or results), and the possibility of filling in with external help - be it human or artificial intelligence. During the development, special attention is paid to eliminating these problems so that the created system would provide a more reliable tool for knowledge-based assessment in the future.

Keywords: multiple choice quiz, web development, educational technology

1. Introduction

The paper investigates a website that is suitable for creating multiple-choice quizzes to test individuals' knowledge. There are quite a few of similar tools, but almost all of them have errors that can be easily exploited, and as a result, the website loses its actual meaning. These problems are:

- Copyable question/answer: Many similar websites today struggle with this problem. It is very easy to trick the system by simply copying the question and solving it in a flash with the help of a web browser or AI. Of course, if we make the text uncopyable, that can also be circumvented, but that requires a little technical knowledge that the average user may not possess.
- Rewrite data with an inspect tool: Everyone has encountered this option in their life, which allows, among other things, editing the code of the website's frontend, be it anyone. Many websites store their data on the frontend, such as which answer is correct or displaying the results. There is no need to elaborate on how much this can take away the meaning of the website.
- Filling out questionnaires with external help: We can hear many stories, even

¹ This article summarizes the author's BSc thesis entitled „Feleletválasztós tesztek készítésére és kitöltésére alkalmas weboldal implementálása” submitted to the University of Miskolc in 2025

from universities, colleges, high schools, or even elementary schools, about someone helping to fill them out in exchange for some kind of payment.

- Use of AI (Artificial Intelligence): The rapid spread of artificial intelligence further reduces the reliability of multiple-choice websites in knowledge assessment. Copy protection can eliminate this to some extent, but nowadays many browsers use built-in M.I. or have image analyzers, so this solution is also becoming less and less reliable.

2. Similar quiz softwares

This section introduces similar software like Kahoot, Quizizz, Google docs.

2.1. Kahoot

Kahoot! [1] is a global educational technology (edtech) platform founded in 2013 in Oslo, Norway. Its goal is to make learning an interactive, playful, and engaging experience. The company's main product is a system that allows users to create and participate in quizzes ("kahoots"). These games are widely used in schools, corporate training, and even for fun at community events. Mission and target audience The founders' main goal was to "Make learning awesome!". To this end, they develop tools that help teachers design interactive lessons, students practice in a fun way, and corporate teams organize experiential training. The platform is used by over 8 million educators and 1.6 billion participants worldwide (as of 2023).

The disadvantages of the site are as follows, according to us: The navigation on it is not the easiest, and it does not solve the above problems. You can copy the questions with answers, if you fill them out once, it is super easy the second time, etc., but the creation of the quizzes itself is very simple and transparent.

2.2. Quizizz

Quizizz [2] is a lesser-known educational platform that enables teachers, students, and corporate teams to create interactive quizzes, learning games, and educational content. It aims to make learning fun and effective through digital tools. The platform allows for the creation of customizable quizzes with real-time feedback, competitive elements (e.g., leaderboards, time limits), and a participant-friendly interface that can be used in classrooms, remote, or hybrid learning environments. Users can not only create their own content, but also access a community content library of quizzes and assignments created by others. Detailed analytics tools allow for tracking student performance, identifying errors, and optimizing processes. The platform also supports collaboration, allowing for group work or integration with other educational systems (e.g., Google Classroom). The service is primarily aimed at educators, students and corporate teams: teachers can create motivating tasks for their subjects, students can practice in a playful way, while companies can conduct interactive training or education. Quizizz is free to use with basic features, but additional options (e.g. advanced analysis, individual branding) are available with a premium subscription. The platform is accessible from both computers and mobile devices, providing flexibility to users.

According to Quizizz's mission, personalized, experience-based learning can be made available to everyone with the help of technology, whether it is school education, self-education or corporate training.

2.3. Google docs (Google Forms)

Most people have probably heard of Google Forms [3] product from Google. It is perhaps the most well-known tool for creating question-and-answer type quizzes, which is perfect not only for knowledge, but also for public opinion polls (it is more often used for the latter). We do not need to register separately to use it (if you have a Google account), but we can just select your account and use it, and unlike the previous ones, there are no subscriptions, so you can use all its features for free. It is completely free, fast, and not only for assessing knowledge. Editing the design is also simple, fast and most importantly: elegant.

Its great advantage is the creation of complex statistics, depicting them with different data using diagrams. Accessing it is also easier than the others, because we can usually simply navigate there through our account.

3. Applied programming technologies

This section presents the technological solutions. It can be divided them into 2 different groups: Frontend (React framework, HTML, Javascript, CSS, and various React packages), and Backend (C#, Microsoft ASP.NET Core, SQLite database, and their packages).

3.1. Frontend

Node Package Manager (npm) [4]: A package manager for Node.js. allows JavaScript developers to easily share and install code packages. The npm Registry is a public repository of open source packages for various applications.

React [5]: Components are the building blocks of React applications, which are self-contained units. A component describes a specific UI element and its behavior (with a .jsx extension), making the resulting code easy to reuse and maintain. It can be best described as object-oriented programming and web development having a common child. This framework is widely supported and has an active community. Development is aided by tools such as React DevTools, which allow you to inspect components and track their status. React documentation is detailed and well-structured, and is available at react.dev. For the project the React Vite is used, which is a faster running/turning version of React. More precisely: Vite is a modern build tool that provides a fast and efficient development experience for web projects. It is becoming increasingly popular among React application developers, as it speeds up the development process and simplifies the project structure. To install it, we need to use node.js, just like in React, with the following command [6]:

```
$ npm create vite@latest my-vue-app -- --template vue
```

Libraries used within react (which had to be installed separately in addition to the Vite packages):

- react-axios: Axios component for React with child function callbacks. Its purpose is to allow rendering asynchronous requests.
- React Router dome: React Router DOM is a browser-specific version of React Router that helps with client-side routing in React applications. This allows us to navigate between different pages without reloading the entire page.

HTML5 (HyperText Markup Language) [7], and CSS3 (Cascading Style Sheet) [8]: HTML is a hierarchical markup language based on SGML syntax, which began to be developed around 2004. This language is used to create the structure and

content of web pages. It is not a programming language, but a tool that allows you to place text, images, links and other elements on a website. It is important to note that it does not have functionality, in order to achieve this you have to use another language (such as JavaScript). The HTML language quite often goes hand in hand with CSS (Cascading Style Sheet). These two are largely the basis of frontend web development, and various frameworks can be built on them, such as Bootstrap for styling, or even together with a programming language, complete website development frameworks can be created, such as React or Angular with JavaScript or TypeScript.

3.2. Backend

The backend handles the storage and execution of server functions, communication with the database(s), and client-server communication with various endpoints.

.NET [9]: A free, open-source, cross-platform development framework from Microsoft. It enables to build modern applications and cloud services on Windows, Linux, and macOS. It supports web, mobile, desktop, game, IoT, and microservices development. It is fast, efficient, and scalable, and has a rich ecosystem, including Visual Studio and Visual Studio Code.

ASP.NET Core [10]: ASP.NET Core is optimized for modern web applications and cloud-hosting scenarios, developed by Microsoft. Its modular design allows applications to use only the features they really need, thereby improving application security and performance while reducing hosting resource requirements. Its official language is C#.

The various packages used these are the followings:

- Bcrypt.NET [11]: It provides password hashing and secure data storage by implementing the BCrypt algorithm. BCrypt is a strong, iterative password hashing algorithm that is used to store passwords and protect them against bruteforce attacks. The algorithm works in such a way that it takes more computing time to decrypt the password, making it more difficult for attackers.
- Microsoft.AspNetCore.Authentication.JwtBearer [12] This is a middleware component designed for ASP.NET Core applications. It enables JSON Web Token (JWT) based authentication, thus providing secure authentication for APIs and web services. This package allows to validate JWT tokens issued by an authentication server, granting access to application resources.
- Microsoft.EntityFrameworkCore [13]: Entity Framework Core (EF Core) is a modern object-relational mapper (ORM) that enables to create a cleanly structured, portable, and high-level data management layer in the .NET (C#) environment.
- Microsoft.EntityFrameworkCore.Sqlite [14]: A database management middleware for Entity Framework Core (EF Core) that enables you to manage SQLite databases in .NET-based applications. The library provides integration between EF Core and SQLite, allowing developers to work with the database using an object-relational mapping (ORM).
- Microsoft.EntityFrameworkCore.Relational [15]: This additional package helps Entity Framework manage database functions (relation, primary key, foreign key, etc.)
- Microsoft.EntityFrameworkCore.Design [16]: It helps with migrations and dynamic DbContext management.

All .NET packages are available on the NuGet website: [17]

4. Design

4.1. Frontend

The frontend is the part of the website that the user directly sees and uses. It determines the appearance and interactive elements of the website, such as buttons, menus, forms and other visual elements. Since it is of paramount importance for the user experience, responsiveness is a primary consideration in frontend development, meaning that the website should be displayed and used well on different devices (desktop, tablet, smartphone).

It is also important to create an intuitive and clean user interface that facilitates easy navigation and quick access to information. When creating my website, we tried to apply the 3-step rule, which means that we can go down a maximum of 3 levels from the main page. The elements of the website are referred as components, because in React this is what the "building blocks" of the website are called.

Components on the diagram:

- Home page: The component that appears to the user when the website (index.html) loads. The central element is where all functions can be accessed from this website, but the most important is filling out and creating questionnaires. It is important that this element is accessible from all components. The header component ensures this.
- Header: The part of the website responsible for quick access to the most important elements. It contains the About page, the users own quizzes (MyQuizes), login and registration, and the button that navigates to the home page.
- Login and registration popup: This is a popup that can be opened from anywhere (i.e. through the header), and here it can be uploaded to the server database or log in to a user account. If this is successful, authentication must occur, for which the JWT token method is used.
- My Quizes: Here we will be able to view the created quizzes, and it can be also deleted. We will also get information about the participants here, since the system can contain two different types of quizzes (infinite, not infinite). Furthermore, here it is able to access the code necessary to complete the quizzes.
- About page: Here we can get the most important information about the website, actually a small advertisement, and after the preview, we can even go on to create quizzes from here, provided that you are logged in.

Creating and editing quizzes: This will be one of the most important elements of the website, here we will be able to create own question sets and modify them. The quizzes will consist of 3 important parts:

- The properties of the quiz, such as the name, how much time is left to complete it, and what data the user will have to provide.
- Creating the questions and answers. There must be a question and answers. It is important to allow multiple correct and incorrect answer options, the best solution for this is a textbox and separation with the ENTER key.
- Creating the members. It will be similar to the questions, except that here the tag value must be included instead of the question. Synonyms can go into the body of the members. In the system we can mark the tag with two special characters: \$animal\$

After these, we need the upload function and a save function, and we also need to check whether the user is currently creating or editing, and the functions must be

adjusted accordingly. It is also important to solve the problem of generating unique codes and checking that everything that can cause errors when filling in is also entered

Filling in: We can get here with the code that we get for the quizzes. First, after entering the code, it is important to enter the data requested by the creator, after which the filling in can begin. How long this takes depends on the type of quiz, which can be the followings:

- If the quiz type is infinite, we go until the first wrong answer is given.
- If the quiz is finite, we go through all the questions once.

Finally, we get our result. In points, if infinite, in percentage, if finite. This data is also immediately uploaded to the database. After that, we navigate back to the main page.

4.2. Website Design

In this subsection, the website design is presented, including illustrating the most important details of the website with diagrams. The website is a simple, transparent, easy-to-use and official look, as it will probably be used more by the 40-60 year old generation.

4.2.1. Important elements of the website

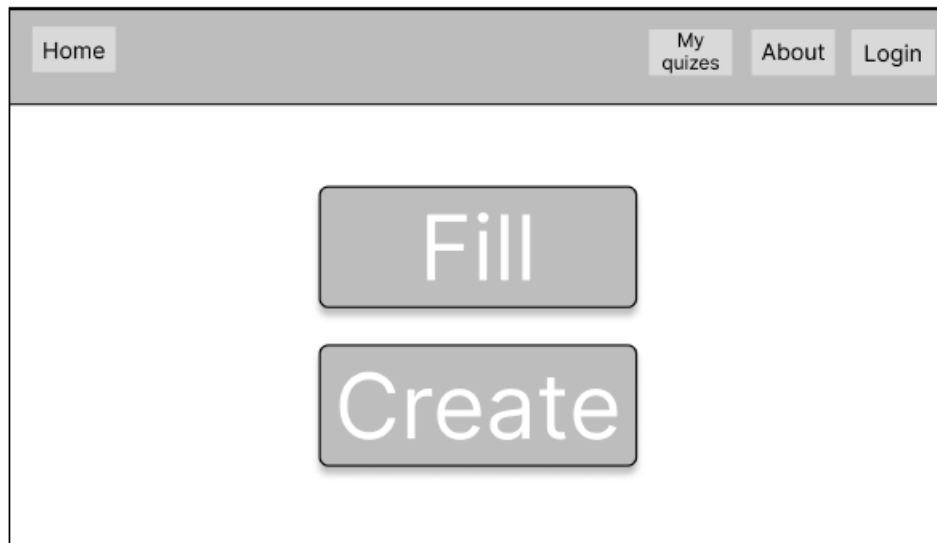


Figure 1. Homepage

The homepage (Figure 1.) of the website is designed to be simple and easy to read. The most important elements of the website (creating and completing the quiz) should be easily accessible. Here, in addition to the homepage, we can also see the header and the elements located on it.

Figure 2. Creation and modification page

Figure 2 shows the draft of the creation menu. On the left we can set the attributes of the quiz: title, time, whether it should be infinite. It can be created using the add button, and in a popup you can specify its name and type. Below are the three controls, the first saves (but not yet to the cloud) our quiz, Discard Changes should be used for modification, because if we don't like the changes, we can discard them all, the third is Publish and apply changes to save to the cloud. On the right are the questions, which can be created with the big + button. The questions will have a header that can be opened (the header contains the question) and contains the answers, and how many answer options there will be. With the Tags button we can add the tags, which will also appear as a popup (at least according to the very first plans).

Figure 3. Adding properties

The Figure 3. will pop up when the user clicks the add button when setting up the quiz data. Here we can add the input fields that the quiz taker must enter. We need a name, which will appear as the caption for the input field, and a type.



Figure 4. Adding tags

These will be placed one below the other, similar to the questions (Figure 4), and we can see them if the user clicks on the `Tags` button. Then a slider will pop up from the left, and they can be added there with an `Add tag` button.

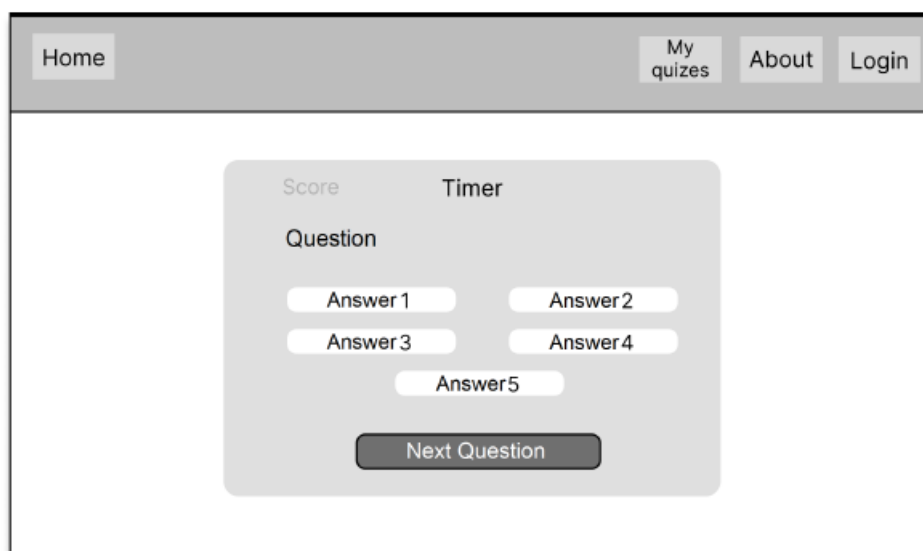


Figure 5. The quiz

Figure 5 shows the filling interface. The score in the upper left corner will only appear in infinite quizzes. The next button will allow us to ask the next question, if we have marked an answer. If the time runs out (which will be measured in minutes), the answer to the remaining questions will be `NULL`.

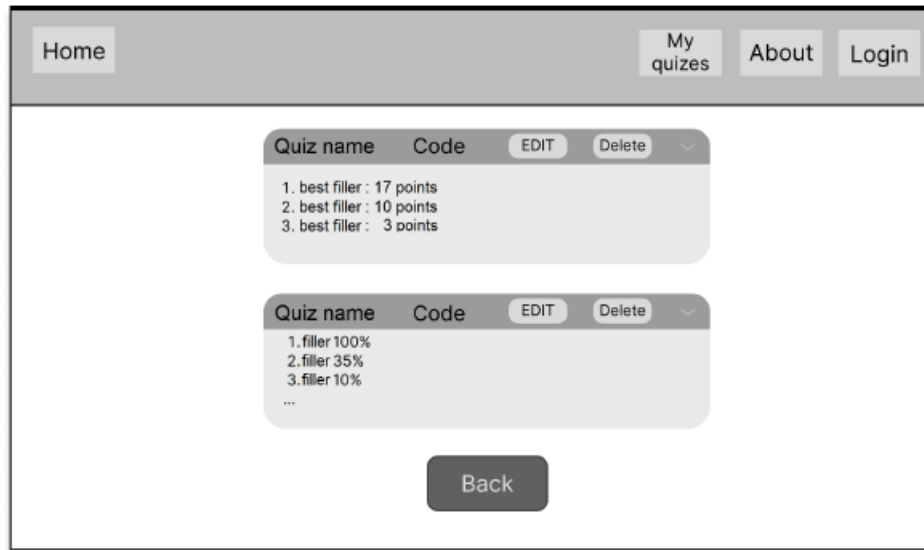


Figure 6. Listing own quizzes

In Figure 6 the interface of our own quizzes can be seen. It can be seen two quizzes here, one infinite (top), the other plain (bottom). The infinite one displays only the first three respondents, while the other displays all of them. The header can be opened and closed in the same way (as with questions) when creating quizzes. The header contains the name of the quiz and the access code, which will consist of random numbers. It also contains the Edit and Delete buttons. The respondents' own data will be visible under the header:

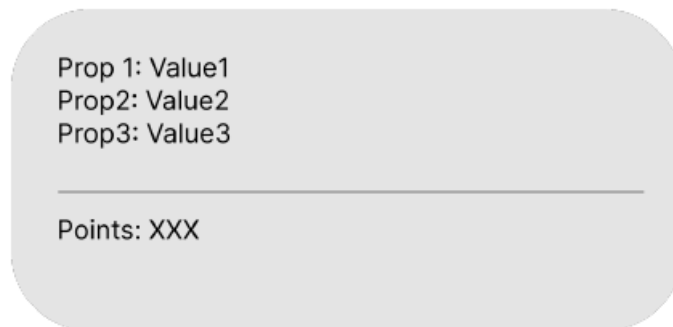


Figure 7. Fill-in example

As shown in Figure 7, the data provided by the user that we requested (Name: Jane Doe, Age: 30) will be displayed, with points or percentages at the bottom depending on the value type.

4.3. Backend

In this chapter, the structure of the backend is presented, along with a more detailed description of the associated endpoints and database.

The backend, also known as the server side, is the part of the application that runs in the background and is not directly visible to users. It includes the servers, databases, and application logic that work together to operate the web application.

- Database interactions: Storing, querying, and manipulating data.

- Server logic: Executing server-side scripts and managing the application's workflows.
- Application workflows: Processing user requests, performing calculations, and returning the appropriate responses.

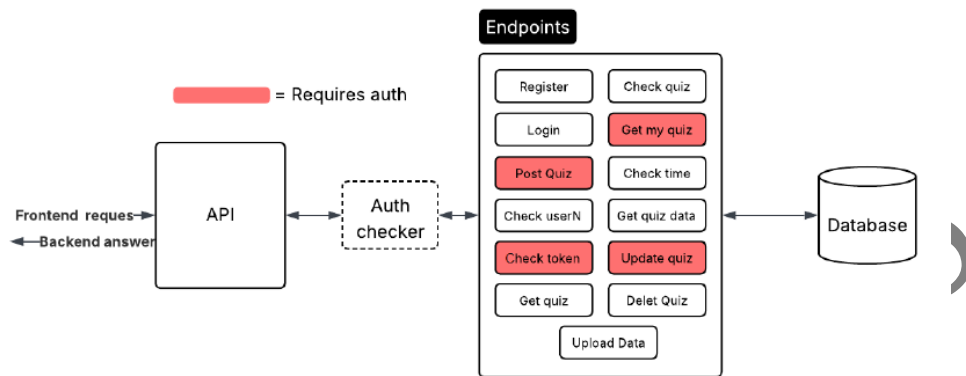


Figure 8. Backend structure

Elements on the diagram are the followings:

- API: This implements the communication between the frontend and the backend, it will choose which endpoint the request should arrive at. In the case of ASP.NET Core, we set it up once and then we don't have to deal with it anymore.
- AuthChecker (Middleware): It will check if the request came with a user account. It is not always active, only if the frontend wants to access the red blocks shown in Figure 8.
- Endpoints: They are the ones that perform the tasks requested by the frontend: they upload data, retrieve it from the database, or check it (some can be accessed at any time, some require authentication).

The description and properties of the endpoints are the followings:

- Register (Method: POST) (Authentication: No): The endpoint required for the user's database. It needs to check if a username already exists, and the password needs to be encoded.
- Login (Method: POST) (Authentication: No): The endpoint required for logging in. It checks the data and returns a JWT token if it is correct.
- Post Quiz (Method: POST) (Authentication: Yes): The endpoint responsible for uploading the newly created quiz.
- check userN (Method: GET) (Authentication: No): The optional endpoint responsible for checking the username match.
- check token (Method: GET) (Authentication: Yes): The endpoint that checks the validity of the token.
- Get quiz (Method: GET) (Authentication: No): The endpoint responsible for retrieving the quiz.
- Check quiz (Method: PUT) (Authentication: No): The endpoint that checks the quiz and stores the data of the quizzes.
- Get my quiz (Method: GET) (Authentication: Yes): The endpoint that retrieves your own quizzes (the quizzes also return).

- Check time (Method: GET) (Authentication: No): The endpoint that monitors the time of completing the quiz.
- get quiz data (Method: GET) (Authentication: No): The endpoint responsible for retrieving the quiz data, without modification.
- Update quiz (Method: PUT) (Authentication: Yes): The endpoint that updates the quizzes.
- Delete Quiz (Method: DELETE) (Authentication: No): Responsible for deleting a given quiz and its data.
- Upload Data (Method: POST) (Authentication: No): When filling begins, a record is created with a start time and user ID.

5. Implementation

This section presents the implementation process. The article describes in detail the more interesting problems that arose during the development and the solutions to them, and the paper will also presents some code snippets that clearly illustrate the chosen approaches. However, it is important to note that the development was not completely linear: some modules and functions were created in parallel or iteratively, so it is possible that a part will return to the description in a later state.

5.1. Frontend implementations

5.1.2. More interesting details

App.jsx: This is the most important file of the frontend, because the index.html file only contains this. It performs all the initialization and comprehensive operations within the program, it is actually the main function. The first and most important function was to ensure that the header never changes, but always stays in place, regardless of which page we are on. Fortunately, this can be solved quite easily in React:

```
<Header loginShowerTrue={loginShowerTrue}/>
..
<Routes>
```

The application has a header and a web page body that changes according to the routes, as if we had two <div/>s. Now let's see what the <Routes> HTML fragment is:

```
<Routes>
<Route path="/" element={<Page/>} />
<Route path="/create" element={<ProtectedCreate/>}/>
<Route path="/fill" element={<Fill />} />
<Route path="/quiz" element={<QuizPalette />} />
<Route path="/list" element={<ProtectedList />} />
<Route path="/About" element={<About />} />
</Routes>
```

This is the router. It is the one that specifies how to access different parts of the web page and condenses them into a route. Anyone can use these routes, as long as they have react-router-dom running on them or use the useNavigate React hook. It is solved the access to the pages required for logging in with own protectedRoute component. This is Create and QuizPalette. They are

the ones that require a token in local storage (`localStorage`).

```
const ProtectedCreate = withAuth(Create);
const ProtectedList = withAuth(ListQuizes);
```

WithAuth: This is an invisible component, whose only task is to protect the protected routes:

```
const withAuth = (WrappedComponent , ) => {
  return () => {
    const token = localStorage.getItem('token');
    if (!token) {
      return <h1>Please login to view this page
    </h1>;
    }
    return <WrappedComponent/>;
  };
};
```

As it can be seen, it gets a component, which it then returns only if the token exists. If not, it displays the warning.

Login and Registration: it is easiest to treat them as a popup entity, rather than as a separate page that can be launched from the header. First, this problem was solved as is typical of popups, the user could exit it even if he clicked out of it. It is solved this by creating an overlay item that only appears if the popup is alive:

```
{showLogin      &&      <div      className="overlay"
onClick={loginShowerFalse}></div><Login
onClose={loginShowerFalse} /></>}
```

Create.jsx: It differs in that the discard changes and Apply Changes are not visible. It is solved these so that they only appear when navigating here from the myQuizes page, It is also used `useNavigate` for this:

```
const modify = () =>{
  nav('/create', {state: {id: quiz.quizID} })
}
...
const id = location.state?.id;
```

slider.jsx, tag_element.jsx: The slider is where the tag elements are stored, and it opens when the tags button is clicked. This was the simplest solution. It would open when the mouse touched a trigger area, but this became annoying after a while.

Finally, the last important problem was to be able to save the quizzes. The elements are saved in `LocalStorage`. The quiz currently preparing calls a simple `SaveData()` function.

```
const SaveData = async() => {
  await signalChange();
```

```

localStorage.setItem('questions',
JSON.stringify(questions));
localStorage.setItem('dataquestions',
....
);

```

However, it is important that before saving the data, we need to retrieve it with a signal. In addition, a function was needed to clean up LocalStorage, which is also called by pressing the Publish and discard changes buttons. The most difficult task with this local saving was that we could edit the quiz, for this we first had to retrieve the aforementioned id variable when loading the website to see if it contained anything.

```

useEffect(() => {
  if (id && !isEdit) {
    setIsEdit(true);
    const fetchData = async () => {
      try {
        const response = await
        fetch(`${API_URL}/users/GetQuizDetails/${id}`,
        {
          method: 'GET',
          ....
        });
      }
    };
  }

```

We had to fetch the quiz, then split the payload we got (quizDetails contains all the data):

```

const transformedData = {
  questions: quizDetails.questions.map(q => ({ id:
q.id })),
  dataquestions: quizDetails.questions.map(q => ({
id: q.id,
...

```

Then place it in the appropriate elements of local storage, and finally reload the page to display the data.

```

localStorage.setItem('questions',
JSON.stringify(transformedData.questions));
localStorage.setItem('dataquestions',
JSON.stringify(transformedData.dataquestions));
...

```

Quizzes: First of all, we had to set up code validation, which is solved by setting the input field to the size of the code (which is 10). As soon as the user submits the code, it is validated in the database, and a response is returned depending on it. If it was incorrect, we indicate it with an alert, if it is correct, we skip to completing the quiz.

```

const submit = async () => {
  const response = await
  fetch(`${API_URL}/users/GetQuiz/${code}`, {

```

```

method: 'GET',
});
if (!response.ok) {
  alert('Invalid code');
} else {
  const quizData = await response.json();
  navigate('/quiz', { state: { quizData, code }
});
}
};

```

If we navigate to the quiz route with the quiz data and the code (this is `quiz_palette.jsx`) this is the middleware that navigates between the data to be entered and the running quiz. First it opens the inputs (this is an `ifSeenProps`) then we go to the quiz. It is important to note that the filling entities on the backend are created when we submit the data:

```

useEffect(() => {
  if (seenProps) {
    console.log(quizData)
    const uploadData = async () => {
      const payload = {
        QuizID: quizData.id,
        Props: propsValues,
        Start: new Date().toISOString()
      }
    }
  }
});

```

There were several problems to solve when completing the quiz, for example inserting the Tags: the frontend would replace the synonyms and separate them because we didn't want to overload the backend.

```

const replaceTags = (text) => {
  const replacements = {};
  return text.replace(/\$([\^\$]+)\$/g, (_, tagId) => {
    if (!replacements[tagId]) {
      const tag = quizData.tags.find(t => t.id === tagId);
      const options = tag?.value.split(",").map(s => s.trim());
      replacements[tagId] =
        options?.[Math.floor(Math.random() * options.length)];
    }
    return replacements[tagId];
  });
};

```

5.2. Backend implementations

ASP.NET Core works like any other web development framework. We need a main startup script where we can initialize its most important settings, currently it is `Program.cs`. The entire server is separate from the backend, so we have to

start both manually (dotnet run).

5.2.1. Database management implementations

The database management is solved indirectly with `EntityFramework.Core`. It works by modeling the database structure in an object-oriented way, i.e. the database tables and their fields appear as classes and properties in the code. It allows us to work with these objects as if they were just plain C# classes, while automatically executing the necessary SQL queries in the background. For example, inserting a new record or modifying an existing data can be done with simple object operations, and we don't have to write separate SQL statements

Step 1: First, different entities is created in C# class-like fashion:

```
namespace Server_2_0.Entities;
public class UserEntity
{
    public int Id { get; set; }
    public required string UserName { get; set; }
    public required string Password { get; set; }
}
```

Creating relationships and foreign keys is similar to other database managers:

```
public int UserId { get; set; }
public UserEntity? User { get; set; }
```

So we need to create a `USER` first to create this entity (this is found in the quiz entity.)

Step 2: Creating `DbContext`: This is specific to the Entity Framework. It is the one that maintains the connection between the database and the application, creating it is not that difficult, just a bit time-consuming:

```
public class QuizWebContext : DbContext
{
    public QuizWebContext(DbContextOptions
    <QuizWebContext > options)
    : base(options) { }
    public DbSet <QuizEntity > Quizes => Set<QuizEntity
    >();
    ...
}
```

Step 3: Creating a migration: This is an optional step to manage the database, but it makes it much easier. It is a version control mechanism that helps us easily change data, tables, entities, etc. We can see the previous changes and what has changed.

5.2.2. Database Management

The database operation must be managed through the `EntityFrame: DbContext`. We can retrieve data, upload data, or create new entities with the new keyword. It is important to issue the `await db.SaveChangesAsync()`

command after each modification. Creating a new entity:

```
UserEntity newUser = new()  
{  
    UserName = user.UserName ,  
    Password = hashedPassword  
};  
db.Users.Add(newUser);  
await db.SaveChangesAsync();
```

6. Summary

The development work resulted in a fully functional website, which in my opinion would also hold its own in the world of the Internet. A significant part of the objectives and problems formulated at the beginning of the project were successfully solved, so the site is not only functionally correct, but also easy to use from a user perspective. The primary consideration when designing the interface was clarity and transparency, but it must be admitted that the visual implementation cannot be considered outstanding. The reason for this is that we do not yet have sufficient experience in the field of web design. Despite this, the interface remained logical, well-structured and readable, which is an important factor in terms of user experience. Overall, however, it can be said that the functionality of the website is fully satisfactory. The solutions it offers are unique and innovative, especially in the area of quiz functionality. We have not come across any application on the market that would address this problem area in a similar way, so the project represents individual value.

However the system has some development opportunities, which are the followings:

1. Public quizzes: quizzes should be accessible not only with code, but also publicly. This can also be requested from Fill.
2. Application of mathematical formulas: currently the website is not able to display calculation formulas aesthetically. This could be remedied. React has several packages for this purpose, but they are not dynamic, and making them so is not a small task, but not impossible.
3. Graphs: The data of our own quizzes should not only consist of seeing the score, but also be displayed with different graphs (like Google Forms)
5. Timing per question: Not only the entire quiz can be timed, but also the questions separately. This way we can do even better against cheating.

References

- [1] Kahoot! <https://kahoot.com>
- [2] Quizizz, Ankit and Deepak. Quizizz – learning games & assessments: <https://quizizz.com>
- [3] Google forms: <https://workspace.google.com/intl/hu/products/forms/>.
- [4] npm (node package manager): <https://www.npmjs.com/about>
- [5] React, Facebook. React – the library for web and native user interfaces: <https://react.dev>,
- [6] React VITE, Facebook. React – The Library for Web and Native User Interfaces: <https://vite.dev/guide/>

- [7] HTML: <https://www.w3schools.com/html/>
- [8] CSS: <https://www.w3schools.com/css/>
- [9] .NET: <https://dotnet.microsoft.com/en-us/learn>
- [10] Steve "ardalis" Smith. Architecting modern web applications with asp.net core and microsoft azure. page 1, 2023.
- [11] Bcrypt.net: <https://www.nuget.org/packages/BCrypt.Net-Next>.
- [12] Jwtbearer:
<https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.JwtBearer/>
- [13] Entityframework.core.
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore>
- [14] Entityframework.core.sqlite.
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite/>
- [15] Entityframework.core.relational.
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Relational>
- [16] Entityframework.core.design.
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Design>
- [17] NuGET galery. .net packages. <https://www.nuget.org/packages>