*Readme (SquizGame) :*

*Malvin NOËL CDOF5*

**Introduction:**

So my web application is called SquizGame. It's a learning app where users can practice on a bank of questions. They'll be able to choose between four answers, and only one of them will be correct. They'll be able to evaluate their speed and the number of correct answers. Users can also access their account statistics. They will be able to see how many questions they have answered, and their average response time.

Normally, the project was supposed to be a 4-person affair, but one group member withdrew 3 days beforehand because two other group members weren't working at all. Namely, the project had not been started. I was with Adam NASSIR, Louis-Melchior and Adrien MONTREER before the group disbanded.

So I decided to make my own project in 3 days. I hope you can take this into account in my rating.

**Database :**

So the database will be on postgreSQL. There will be five tables: Answers, Questions, Thames, UserStats and Users.

A screenshot of a computer

Description automatically generated

The database will be set up on PGAdmin 4, where we've created a Squeeze Game database with a Squeeze Game User, enabling us to connect it to the back-end.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedTo insert data, we'll use PGAdmin's query tool. We'll then be able to create tables and question banks.

**Backend :**

A screenshot of a computer program

Description automatically generated

For the back-end, I used Node.js and Express as well as TypeScript. We created two different folders: a source folder where all the TypeScript files will be stored, and the build folder where the Node.js files will be created automatically.

To synchronize the back-end and front-end, I'm going to use an advanced web feature, WebSocket.

The SQLize file will enable the back-end to synchronize the database with the data received from the front-end.

When the user enters his password, it will be encrypted in the database using the Bécrypt library. This protects the user's privacy :

A screen shot of a computer screen

Description automatically generated

The back-end is connected to the database:

A screen shot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generatedFrontend :**

For the front-end, I used Angular 17 and ng-bootstrap. I then created various components. The dashboard for the user, a home page for the game's home screen, a screen for playing, a screen for registering, a screen for this login, and the user will have access to statistics and traditional games. The dashboard for the user, a home page for the game's home screen, a screen for this login, and the user will have access to statistics.

The problem was that Angular 17 is very recent and documentation is still sparse. That's why the code worked for a while, but after the switch to Angular 17, it didn't work anymore. The connection with the back-end via WebSocket was no longer working.

**The Interface via the frontend :**

When the user logs in, he or she is directed to the game's home interface. This interface corresponds to the front-end on-page file.

A screenshot of a login form

Description automatically generated

Then, if you already have an account, you can log in by clicking on Login.

A screenshot of a login page

Description automatically generated

On the other hand, if the user doesn't already have an account, this will be automatically detected on the back-end with the WebSocket sent from the front-end to the back-end. And the back-end will be able to detect whether or not the user has already registered. If not, he'll be automatically redirected to the interface to register.

A screenshot of a login form

Description automatically generated

The user can then register on the application. This will send the information to the back-end with the WebSocket, which will synchronize the database with the new information.

Once logged in, the user lands on the Dashboard interface:

So he can choose whether he wants to play or see his stats.

A screenshot of a dashboard

Description automatically generated

When the user clicks on Play, First, he can select a term from these four themes.

A screenshot of a computer

Description automatically generated

After, he'll be able to play the question-and-answer game, with a series of 10 questions in which points will be calculated according to the user's response speed, as well as whether he answers correctly or incorrectly.

A screenshot of a video game

Description automatically generated

When the game is over, the score is displayed. And the user can start a new game.

A screenshot of a video game

Description automatically generated

When the user clicks on statistics of its dashboard, he'll have access to various statistics from the first time he played the game.

A screen shot of a game

Description automatically generated

**Conclusion :**

On this project, I managed to create a database and connect it to the back-end. One of the main obstacles I encountered was establishing a connection between the frontend and backend using WebSocket. This proved particularly difficult due to the new functionality of standalone modules introduced in Angular 17. At first, I wanted to use HttpClient to connect the backend and frontend, but it didn't work at all with these new modules.

Standalone modules represent a significant departure from Angular's traditional module system, offering a more simplified way of organizing and priming components. However, it also means rethinking conventional methods of integrating services such as WebSocket.

Nevertheless, I was able to propose a functional front-end and a back-end connected to the functional nose drop too.