SeQG - A Security Question Generator as a Webapplication

Project Documentation
of

@Finn Kock, @Alejandro Castilla,
@Sergio Pajuelo , @Robert Eggert

Ludwig-Maximilians-Universität München
Practical Course Usable Security

Reviewer:	Dr. Viktorija Paneva and Doruntina Murtezaj
Professor:	Prof. Florian Alt

30.07.2025

1 Index

2 Introduction

This is the intro to our project

3 Frontend

this will get split up later

4 Backend

The Backend is the silent core of the whole project, which makes it possible to create the connection between the user and front-end. As the user establishes a connection to the front-end the next Backend component is in use, namely the interaction between the front-end and the LLM API After a session our backend then also covers the safe disconnection logic and resets everything to a clean state.

4.1 QR Code - Connection

The whole connections starts by making sure the user visits the link via. scanning the QR code. Afterwards a link gets generated, that is unique for every session and contains a JWT token. This token gets generated by the backend and send over to the front-end and only then the user may connect. In that way, we make sure that no connection can be established without the user having the right to do so. The front-end then uses this exact JWT token and only allows connections with the same token in current use. Considering all previous steps, the actual connection logic is then handled by the users front-end, from which the device (e.g. phone) emits a ”register” message to the backend. This leads to the final step of the connection, where the backend receives the message and sends over the status that is appropriate for the current connection state. If everything worked optimally the front-end gets the message ”connected” emited from the backend.

4.1.1 Role & Status

First we have to define what actual roles can connect, this makes random or brute connections even harder. Currently we have two roles, namely the ”client” and ”host” role. The client is the one who connects to the host, whereas the host is the one who gives the opportunity connecting to the currently active session.

To make sure a connection is actually valid we now discuss the part where we need to define different states helping us understanding what went pottentially wrong in a connection.
The following states currently in use are defined as:

- **pending** - The QR Code is created and the front-end is waiting for the user to connect.
- **connected** - The user has successfully connected to the session and can now use the application as intended.
- **disconnected** - The user has disconnected from the session, either by closing the application or by a timeout.
- **already_connected** - The user or a host was already connected to the backend, which moves potential intruders into a url sinkhole.

The states that should be added for a more detailed error handling are:

- **no_host** - The user tries to connect to a session that does not exist.
- **invalid_role** - The user tries to connect with a role that does not exist.

4.1.2 Registration

Making the connection valid though it is not enough to just scan the qr code. We actually need a registration logic that makes sure that the host and client are allowed to connect. This is done by the host front-end and clients front-end, who both need to emit a ”register” message to the backend. By registering, the backend recognizes if a host or client is already connected. This makes flag handling easier, as the backend can then emit a message to the host or client front-end appropriate to the actual status.

4.2 API Calls

In actuality currently there exist four API calls the qr-code backend is handling.

- **/connect/host** - This is the call that is used by the host to connect to the backend and receiving the JWT token.
- **/disconnect/all** - This call is used by the client or host to close the current session.
- **/private/user-data/:userId** - Here we get the private-user data from the backend, which is already stored (currently only age and experience).
- **/private/saveAgeAndExprience** - This call is used to save the age and experience of the user in a private session.

Adding more API calls is possible, but currently not needed. If one shall adjust those calls then only in the matter of better verification, so that no workaround allows to use these API calls without the right to do so.

4.3 LLM - Connection

Before the user gets to actually answer questions provided by the LLM, we need to submit our age and experience, if we did not already do so. Only then the questions will start to get fetched from the LLM API. But this is not happening directly, for that we also implemented a little backend, that needs to be started, which then handles all needed API calls. Currently though the backend is not protected against calls from outside, so it is not recommended to run it on a public server. One might first want to implement some kind of authentication, before doing so. The same goes for the LLM API, as it is a Ollama Backend that runns openly. Here we also recommend to implement authentication procedures such as private keys or doing network segmentation only for the API to prevent unwanted access.

4.3.1 Tips

...

4.3.2 Questions

Here we differentiate between the two modes as one of them has a userId and the other one does not. Considering that, lets first look at the guest-mode question api request. For that we have prepared a prompt and a list of topics from which the llm can choose. Both of them are stored in the backend folder. From here on we directly pass arguments into the prompt such as age, experience and the topic. When we are done, we simply send over the whole prompt to the LLM API, which then responds with a specific json format. The json format is then parsed and send over to the host front-end. Formatted the json represents this structure:

```
{
    "question": ... ,
    "option_s": ... ,
    "dropZones": ... ,
    "correctAnswer_s": ... ,
    "topic": ... ,
    "questionType": ...
}
```

Each one having there own meaning.

- **question**: The question that the user has to answer (String).
- **option_s**: The options that the user can choose from (String array).
- **dropZones**: Represent drop zones in a drag & drop event, where the user can drag and drop the options (String array).
- **correctAnswer_s**: Correct answers for the question (String array).
- **topic**: The overall topic of the question (String).
- **questionType**: The event-type of the question (String).

4.3.3 Explanation

...

4.3.4 Saving Answers

...

4.3.5 Feedback

...

4.4 Anti-Breaching Sessions

4.5 File Cleansing

4.6 Contributors

@Finn Kock: ...
@Alejandro Castilla: ...
@Sergio Pajuelo: ...
@Robert Eggert: ...