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

Index

1 Introduction

In this day and age, having a solid understanding of cybersecurity fundamentals is crucial in everyday life. These threats aren't the problem of just a few skilled people anymore, but a common crime that unfortunately affects hundreds of thousands of people every day.

Just to provide a glimpse of the magnitude of the problem, the Federal Bureau of Investigation's Internet Crime Complaint Center (IC3) releases their annual report, in which it measures the financial losses due to these crimes. In 2024, the report estimated a total loss of $16 billion dollars (a 33% increase compared to 2023). These staggering losses highlight the necessity of cybersecurity awareness in every day life.

When it comes to educating users in cybersecurity, the same problem often arises: it is almost impossible to achieve a teaching difficulty level that suits everyone. The one-size-fit-all approach makes it unattractive for users to spend their time learning, as they often find the content too easy or too hard.

To help solve this problem, we used the innovative LLM technology, which enables us to personalize the user's entire experience (from questions to personalized feedback). The main idea behind SeQG, is to take into account several factors (depending on the mode selected by the user), such as age and cybersecurity experience. Based on this self-assessment, the LLM will fetch questions according to the user's level.

In the Private Mode, in the LLM prompt we also include the user's performance on previous cybersecurity topics. This allows the LLM to fetch more questions about the topics, where the user's knowlegde lacks, helping them make progress.

By combining all these factors, we achieve an approaching optimal learning process in a gamified way (using several question types), where we can adjust the difficulty level that best fits the user.

2 Frontend

Frontend definition

2.1 General Setup

The SeQG frontend is built as a modern web application, with React and TypeScript to ensure a smooth and responsive user experience. It is structured in a way that supports scalability, clarity, and ease of development. The overall design of the application is split into smaller and reusable components. Each part of the interface, from full pages to small buttons, is organized on purpose. This simplifies maintenance and the navigation through the codebase over time. Visually, the interface is designed using Material UI (MUI) this ensures everything looks consistent, works well across different devices and screen sizes. Buttons, layout elements, and interactive features follow the same design language. Altogether, the frontend setup ensures a fast, flexible and user-friendly experience, while keeping the system lightweight and efficient.

2.2 File Based Routing

2.3 Navigation Screens

Until a user approaches and interacts with the display **security tips** are getting dynamically fetched by the LLM and shown on the screen. These messages, shown in Figure 2.1, are designed to catch user's attention and encourage them to engage with our **SeQG** and learn more about **cybersecurity**. On touch interaction the user gets navigated to the **mode selection** screen.

Figure 2.1: Main Screen

The **mode selection** allows the user to choose between one of **two mode characters**, each accompanied with a short explanation of their functionality, as to be seen in Figure 2.3

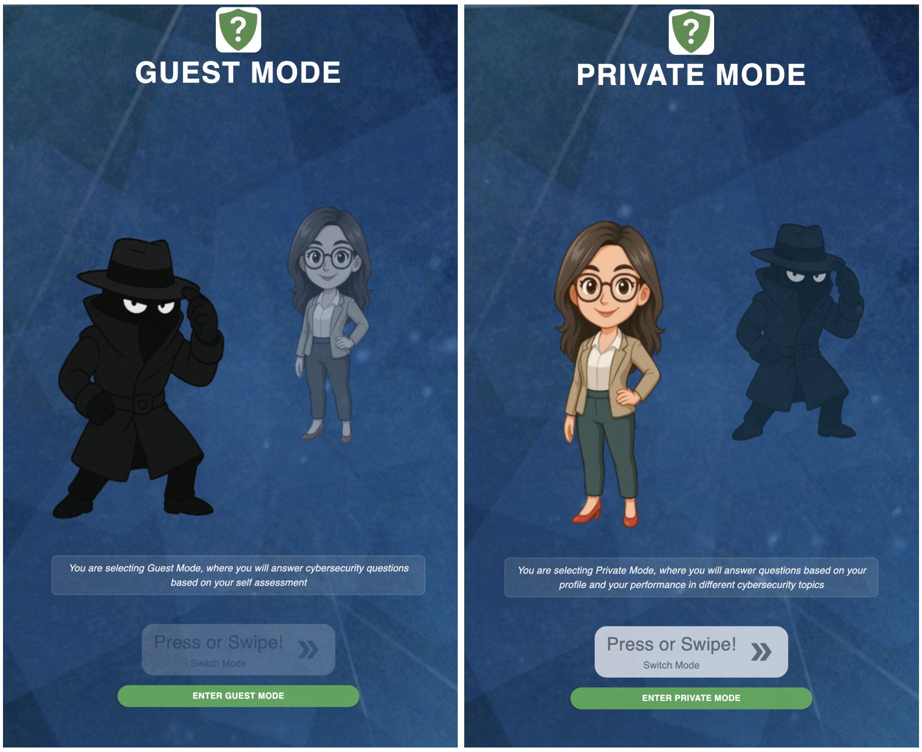


Figure 2.2: Characters for Mode selection

2.4 Welcome Screens

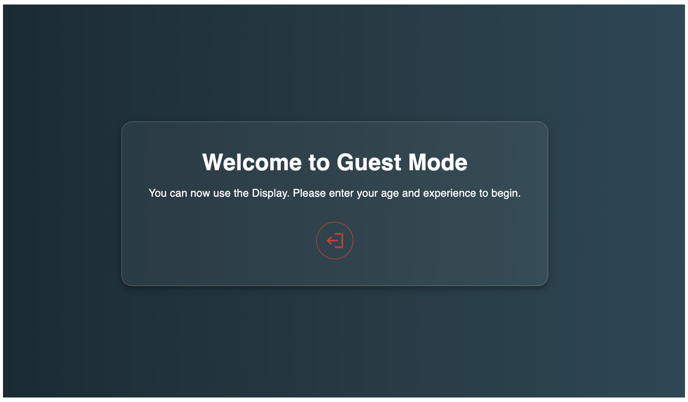


Figure 2.3: Connected as Guest-User example

When the user scans the QR code with their device, a screen will appear allowing them to leave the session remotely without needing to interact directly with the public display. If the user chooses the private mode, a **session token** linked to the user's device is stored server-sided, preventing other user's from interfering with the same session.

Figure 2.4: Forbidden connection example

If a user is already connected to the server, any subsequent user scanning the QR code will see a different screen on their device, informing them, that the **session is currently occupied**.

2.5 Modes

2.5.1 QR Code - Integration

QR codes play a central role in how devices connect within **SeQG**. They are used as a simple and intuitive way to link a user's device to an active session. When a session is started, the application generates a QR code containing a unique link. This link includes a hidden session identifier that tells the system, which session the user can connect to. Once scanned, the device opens the link, which automatically redirects the user to the corresponding session within the app. The beauty of this process is that it requires almost no effort from the user, as no code or password is required. Everything happens in just a few seconds by scanning the QR code.

2.5.2 Communication with Backend

The **SeQG** frontend communicates with the backend through WebSockets to enable real-time, bidirectional data exchange. Here we differentiate between client and host frontend. The host frontend must stay in constant connection with the backend, while a session is active, whereas the client frontend needs to connect only once. Before the client can connect, the host frontend must first request a JWT token and then connect to the backend with this token. As the user joins a session, the client's frontend uses the previously fetched JWT token from the host's frontend and connects to the backend, via the URL saved in the QR Code. While connecting, the client's frontend emits an ”register” event containing:

- the token from the URL
- a role
- the user's unique ID from localStorage (if private-mode)

If the session is valid and the user is allowed to join, they're granted access and the interaction begins. If not, they receive a ”already_connected” message, which can then be handled as an invalid user. This approach makes it easy to manage users in real time.

2.5.3 Guest Mode

Guest Mode is designed for quick, one-time participation without any setup or login. It allows users to access a session instantly without any user data being stored. When someone joins a session in Guest Mode, the application automatically generates a temporary user identity and stores it locally in the browser. This identity helps the system recognize the user during the session. Guest users have full access to participate in the session's activities, but their presence is temporary and not tied to any personal information. Once the session ends or the browser is closed, the user effectively disappears from the system's memory. This mode is ideal for situations where users are not expected to return.

2.5.4 Private Mode

Private Mode offers a more structured and secure way to connect to a session. It is used when a user scans a QR code that links to a specific, private session. This link contains a hidden token that identifies the session and grants access to it. When the user opens the link, the application uses this token to establish a secure connection with the backend system. At the same time, the user's device is assigned a unique identifier, which is stored locally and remains available for future sessions on the same device. This allows the system to recognize returning users, maintain a consistent identity across multiple sessions, and apply session-specific rules or roles. The connection is verified before granting access — so users without the correct token or link will be blocked.

2.5.5 Question Types

SeQG has **six question types implemented**, enhancing the user's gamified experience with a dynamic, entertaining app. Each of these question types has been implemented with visual animations depending on the correctness of the answer and each one of them has an explanation generated by the LLM in case the user answered incorrectly. The six question types are:

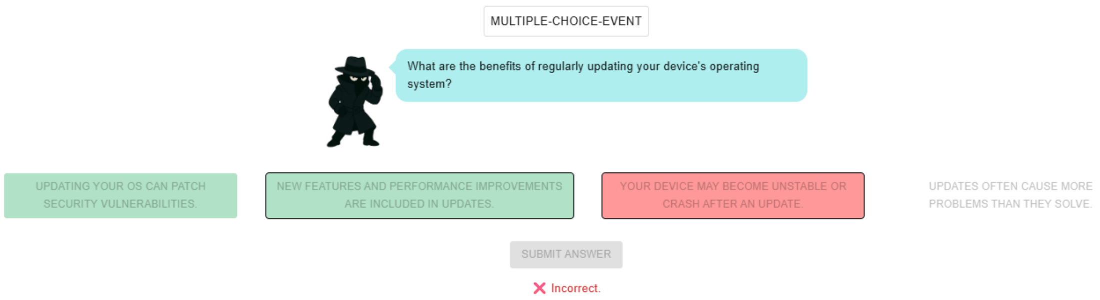
- **Single choice**
- **Multiple choice**
- **Think event**
- **Drag & drop**
- **Sorting**
- **Line-connect**

Single choice

In this question type, the user is offered **two possible solutions** and has to select the correct answer. Once they touch on the desired answer, the correct answer will be highighted in green. If they have failed, an instant LLM explanation will be fetched on the same screen for the user to learn:

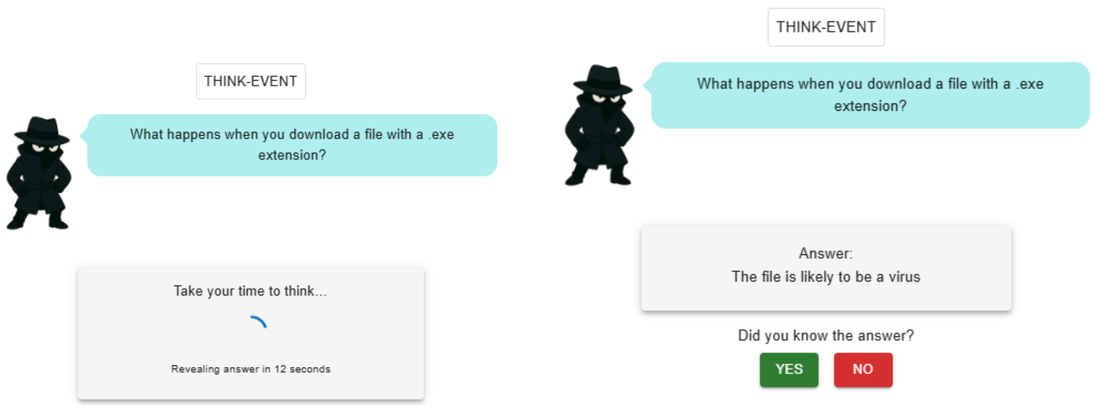
Multiple choice

Here the user is given **between two and four possible solutions** and has to select **all** correct answers. Once every correct asnwer is selected, the must press the submit button:



Think Event

In the think event, the user must think about the possible answer to the question formulated for **fifteen seconds**. Once the time is up, they must (honestly) select whether they knew the correct answer or not:

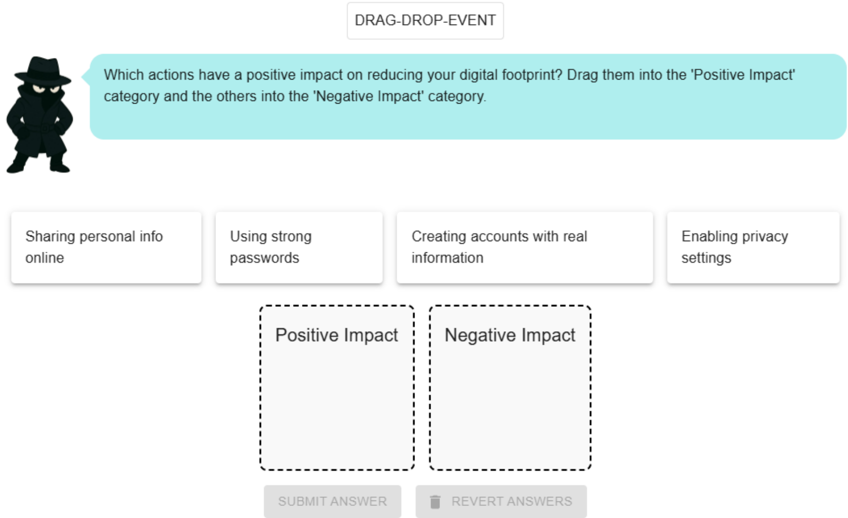


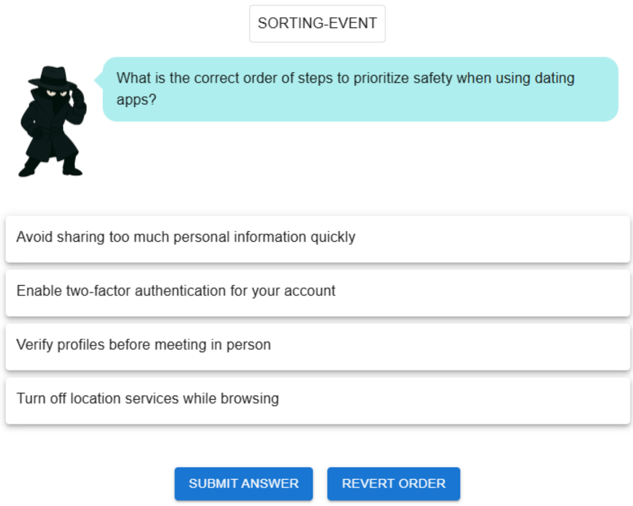
Drag & Drop

The user must drag the boxes into the **correct category**. There is also the possibility to revert the answer:

Sorting

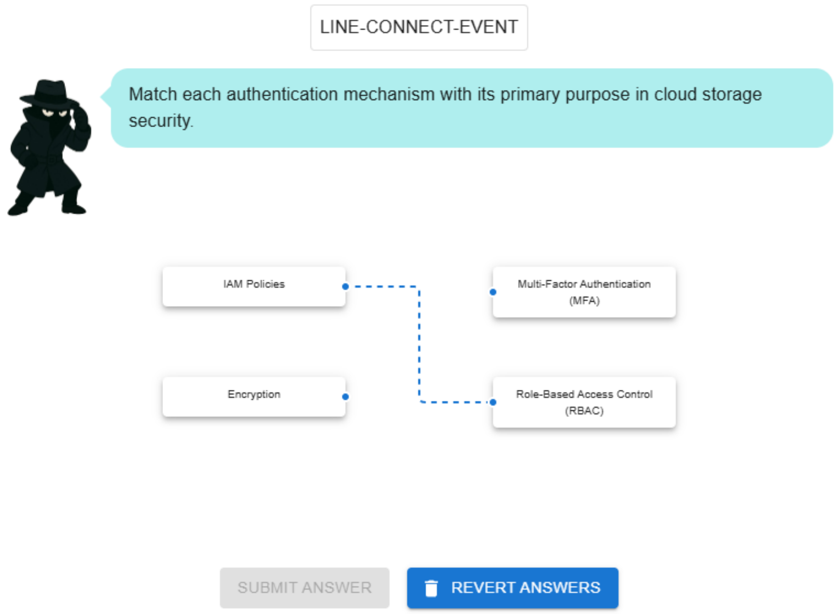
All the answers must be **correctly sorted**. As before, the user can return to the original order whenever they want:


DRAG-DROP-EVENT
Which actions have a positive impact on reducing your digital footprint? Drag them into the 'Positive Impact' category and the others into the 'Negative Impact' category.
Sharing personal info online
Using strong passwords
Creating accounts with real information
Enabling privacy settings
Positive Impact
Negative Impact
SUBMIT ANSWER
REVERT ANSWERS


SORTING-EVENT
What is the correct order of steps to prioritize safety when using dating apps?
Avoid sharing too much personal information quickly
Enable two-factor authentication for your account
Verify profiles before meeting in person
Turn off location services while browsing
SUBMIT ANSWER
REVERT ORDER

Line-connect

In tis question type **an even number** of boxes is displayed, having to match each item on one side to their correspondant on the other side. There is also the answer reversal button to start again:



2.5.6 Gradings

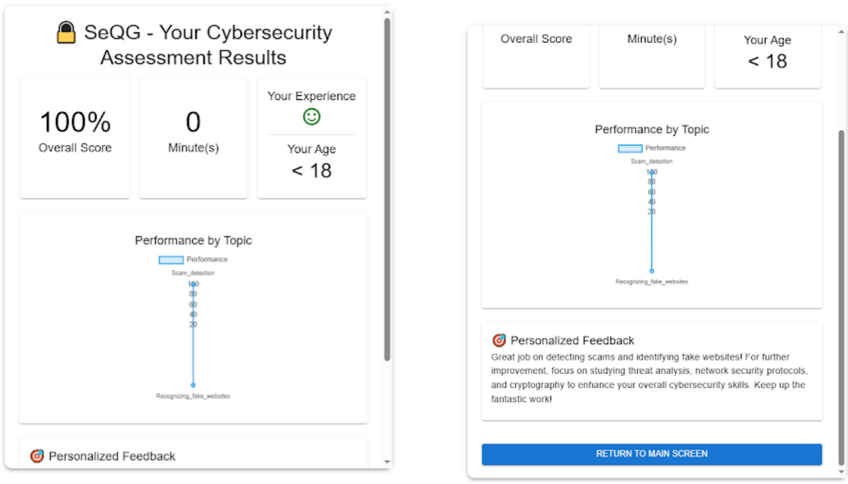


Figure 2.5: Dashboard

When the user finishes their session (in both guest and private modes), a dashboard appears summarizing their **performance**. To enhance user feedback and engagement in SeQG, the dashboard includes radar charts assessing the user's performance across each specific **cybersecurity topic**, generated with Chart.js. The diagram is split into two if there are more than eight topics, to avoid exceeding the visual limit. The dashboard also includes **personalized feedback**, as shown in Figure 2.5, which highlights the user's strengths and weaker areas during their session, encouraging them to use SeQG again to improve their **cybersecurity awareness**. User Progress is persistently stored in a structured JSON format if the user started a private session, including metadata such as their userID, age, experience and per-topic experience.

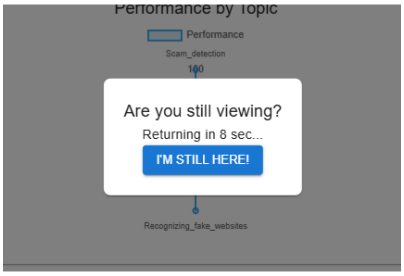


Figure 2.6: Inactivity timer pop-up message

The interface also includes an inactivity timer that triggers a message in the center of the screen after 60 seconds of **no interaction**. The user then has 10 seconds to confirm their presence; otherwise, the session is closed and the system returns to the main screen.

3 Backend

The backend is the silent core of the entire project, which makes it possible enabling connection between the user and front end. As the user establishes a connection to the front end, the next Backend component is made active, 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.

3.1 QR Code - Connection

The connection 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, which contains a JWT token. This token gets generated by the backend and send over to the front end. Only then may the user 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.

3.1.1 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.

3.1.2 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. If the user is in private mode, then we emit a special message called ”register-private”, which then creates a file with a unique uuid and a token in the local storage of the users browser. This file and local storage token is only created though, if the user submits his age and experience first. Such a file would then look like this:

```
{
    "user_id": "abc123",
    "age": "< 18",
    "experience": 2,
    "progress": {}
}
```

Where the **user_id** (String) is the unique identifier for the user, **age** (String) is the apporximate age of the user, **experience** (Integer) is the experience level of the user and **progress** (Object) is an object that will hold the progress of the user with counted questionTypes and if they where answered correctly.

3.2 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.

3.3 LLM - Connection

Before the user gets to actually answer questions provided by the LLM, he needs to submit his age and experience, if not already done. Only then the questions will start to get fetched from the LLM API. But this is not happening directly, for which 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.

3.3.1 Tips

To catch a potential users attention, we decided to use Tips and eye catching messages on our main screen. This shall then make the users interest rise about our PSUI and let him or her approach our Public Display. These tips are getting fetched from the LLM API. In case of server connection issue, we have a fallback mechanism that uses locally saved tips. Later on in the future, one may also add not only tips, but also current news in cybersecurity and generally speaking of security awareness. There are multiple API available, even for free, that allow to fetch current events in cybersecurity.

3.3.2 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. The formatted **json** respond then, 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).

3.3.3 Explanation

In the explanation we actually dont need to differentiate between the two modes, as the explanation should be universal. There may be a critical moment where one might think to add some extra explanation features for the private mode, but we did not implement that. As we have fetched a question from the LLM API we can refeed it with almost the same provided json format. In our case we only need the question and the correctAnswer_s aswell as a boolean telling the LLM we did indeed answer the question incorrectly.
When we do answer correctly there is no need to explain the question, as the user already knew the answer (hopefully did not guess). This means we have a simple return format for our explanation, because in the end we only need the explanation text.
Therefore our **json** format looks like this:

```
{
    "explanation": ...
}
```

Where the **explanation** (String) is the text that the LLM provides us with.

3.3.4 Saving Answers

When the user enters the guest mode, we do not save anything but rather do a local session with grading at the end. However, when the user enters the private mode, we do save the answers, so that his or her previously answered questions can be taken into account. This makes the LLM way more personalized and the user can also see his or her progress eventually. The main idea is, that we get the Questiontype from the LLM API. After that we add to our autogenerated file with the userId as its filename, the current Questiontype. For the final part, we increase a counter for the correctly answered questions according to the Questiontype. If the user indeed answered the question correctly, we increment by one, otherwise not. The total amount of questions is also saved for each Questiontype. This amount always gets incremented by one, no matter the correctness of the answer.
In general a progress output may look like this:

```
{
    ... ,
    "progress": {
        "passwords": {
            "correct": 0,
            "total": 1
```

```
        },
        "camera_microphone_access": {
            "correct": 0,
            "total": 1
        },
        "kids_online_safety": {
            "correct": 2,
            "total": 2
        }
    }
}
```

Which would mean that in total the user has answered 4 questions, of which he answered 2 correctly.

3.3.5 Feedback

The moment the user decides that he or she has had enough of the questions, one can request a feedback. This feedback is then generated by the LLM API and sent over to the front end. The user has now the oppertunity to read the feedback and then decide for himself learning more about a topic or not. Feedbackwise we offer a spider chart limited to the maximum amount of 8 topics, where we define from 0 to a 100 percent how well the user did in each topic. Aswell as a specificly generated text, that gives the user a more detailed overview of his or her performance. Afterwards the user can disconnect from the session or do nothing as the session will automatically disconnect after a certain amount of time.

3.4 Anti-Breaching Sessions

As every server, that is reachable from the internet or inside a local network, we need to make sure, that we are as secure as possible. For that we implemented a simple anti-breaching session logic. Everytime a user connects to the backend we check if someone is already connected (max amount of client: 1, host: 1). Therefore moving the unwanted user into a url sinkhole. This means that refreshing the page will keep you on a prohibited url where one cannot do anything, which eases the backend servers capacity. Additionally to that we have a structure, where even if an attacker gets the information about the current session, he cannot make anything of it. The actual data that is stored from the user does not contain any sensitive information, but rather a unique userId, a rough approximate of the age aswell as the experience level, the user provided us with. Additionally the JWT token is only valid for a certain amount of time, which makes it even harder to breach the session. Though our application is not perfect and we do not claim it to be, we still try to make it as secure as possible. One can for example try to impersonate a user by storing the userId in his or her own local storage, this would mean, that the attacker may use the interface as the pronounced user. However the attacker also has to be motivated to then answer questions for the user, which he or her is impersonating. This has no actual benefit for the attacker. We also do something that is almost bullet proof, so that even if an attacker mages to connect to the session there is nothing the attacker can do. As the QR Code is generated the user scans it. In that Moment the file is being created or reused from the connected user. Even if the attacker tries to connect now he or her will be send to the sinkhole as we remember the first user that connected with his or her specific userId. Therefore making sure the file of the user is safe and not touchable by a potential attacker.

3.5 File Cleansing

As the user logs into the private mode, we create a file, but how do we remove it again? There are alot of ways one could do so, but most of them require some interaction. Therfore we decided that a simple python script, running maybe once a month, shall remove all files that are older than 30 days. This may seem unfortunate for the user as his progress is lost after one month but for the servers memory sake we need a way to remove files after a certain amount of time. If we try to keep them forever this may lead to dead files, as users may never return or users often delete there browser cache, maybe even change there browser. All that has influence on how the files are getting accessed and therefore we really need a way to remove them. The script itself is in the backend folder called "checkFiles.py".

3.6 Contributors

@Finn Kock:

- Creating UI concept for the mode selection and implementing it
  - Animated Background
  - Item orientations in the Mode Screen
  - Character Designs
  - Character Rotations on touch and button click
  - Transparency of character in the back
- Implement Icons as buttons
- Animating text
  - Create multiple character reactions
  - Choosing a random character reaction according to the mode each time a question is asked
  - Putting the question in a question box with a precalculated length
  - Textanimation for Questiontext and Answertext

@Alejandro Castilla:

- Security Tips on the main screen, fetched from the LLM API, with a fallback mechanism in case of server issues.
- End Session screen with performance summary (user statistics and grading) and inactivity handling logic (confirmation pop-up and auto-close).
- Spider chart visualization of user performance in percentages, with dynamic splitting in case the user answers questions on too many different topics.
- Feedback message fetched from the LLM, encouraging users to use SeQG again to improve in their weaker topics.
- Adjustments to several components to ensure proper centering in the demo display.
- Backend integration improvements and JSON parsing corrections in the prompts for reliable communication with the LLM.

@Sergio Pajuelo:

- The whole welcome screen when scanning the QR code, including differentiation between modes.
- Additional information to the user, such as the consent to process their data (with pop-ups), fetching information, etc.
- Each mode's screen after selecting the avatar, including the mode description and animated features, dynamic background, etc.
- Answer highlighting in the questions for visual feedback to the user.
- Improving the user's visibility of the modes descriptions.
- Resizing/redesign of the initial screen components and animations as well as the mode selection one due to problems in the demo display.

@Robert Eggert:

- The Whole backend logic, including the LLM API, QR Code and the Anti-Breaching Sessions idea (except the Feedback api call).
- File based Routing and the File Cleansing logic.
- Most of the State and session management and the Connection/Disconnection logic.
- All question Types + overlay
- Touch compatability
- Configs