LLM-based chatbot for online Escape Rooms

DigiKompASS is funded by the Foundation for Innovation in University Teaching.

LLM-based chatbot for online escape rooms

1. Concept

As part of the GAME-LOAP project, a digital escape room is being created as a test object for use in teaching. The escape room will be created using the learning content creation tools from Lumi Education. Lumi uses the H5P file standard, which can also be packaged as a SCORM package. Based on the openness of the SCORM standard, the Escape Room in this project is extended by an LLM-based chatbot customised with prompt engineering.

2. Architecture

In order to talk about the architecture of the chatbot extension, the architecture of a SCORM package must first be known.

A SCORM package is a ZIP archive of a very simple, exclusively client-based web application. It is therefore a packed archive of HTML, CSS, JS and media files that are downloaded to the user's computer to be executed locally in the browser.

The export of SCORM packages is destructive with regard to changes made to the code. A new export overwrites all changes. To make changes to the code, the learning content must be exported, unpacked, modified and packed again after each change to the learning content.

This project makes the following changes to an existing escape room exported from Lumi Desktop:

1. A function is added to the h5p_adaptor.js file. This function serves as the entry point for all further changes. It is the only change to existing SCORM package files.
2. A folder /Chat is added to the root directory of the SCORM package. This folder contains all other files for the extension (images, code, HTML)

The aim of this structure is to change as little existing code of the SCORM package as possible in order to avoid unnecessary work after updates of the Escape Room. For this reason, it should also be possible to add the other files as a single folder.

With regard to the interface of the escape room, the extension works like an overlay, it is superimposed over existing elements as a new layer. Existing elements are not changed.

In order for the chat interface to function as an LLM chat (in this case ChatGPT), it requires a connection to the OpenAI API. As SCORM packages are exclusively locally executed applications, this API cannot be accessed due to CORS. The solution is a proxy server. As SCORM packages do not use servers, there must be a server somewhere in the connection between the user and the API that has the API key and can reach the API, but has no restrictions on the incoming side. Such a proxy server is provided with this project using Cloudflare Workers.

DigiKompASS is funded by the Foundation for Innovation in University Teaching.

This results in the following architectures:

Figure 1 - Sequence of API calls

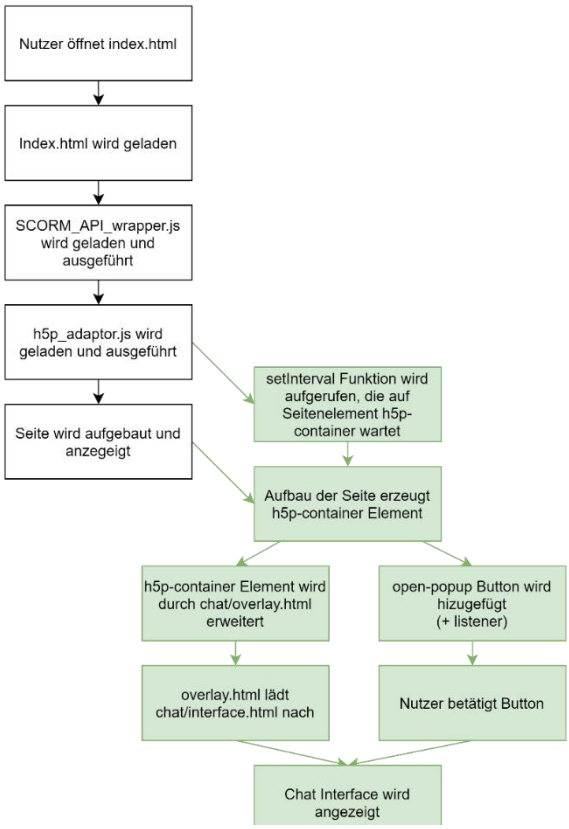

Figure 3 - Page layout with chatbot

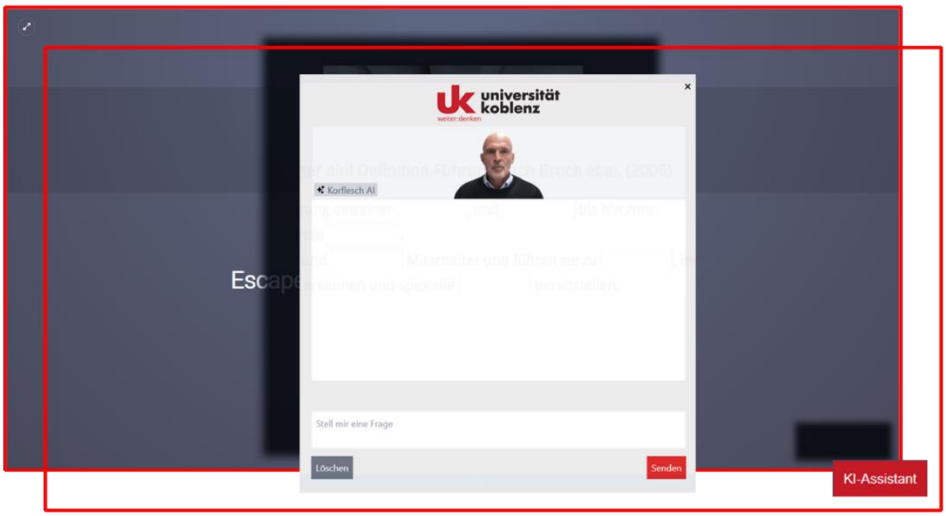

Figure 2 - Page layout,

DigiKompASS is funded by the Foundation for Innovation in University Teaching.

3 Implementation

The actual implementation of the chatbot "plugin" for SCORM packages is divided into three parts. The actual chat interface, the function that connects it to the existing SCORM package, and the proxy server on Cloudflare. In this chapter, I will go into more detail about the exact technical implementation of the individual parts so that the structure can be understood and adapted appropriately if necessary.

3.1 Chat interface

The chat interface itself also consists of two parts. The button that opens a popup and the chat interface itself. The button, including the import for the chat interface, is defined in Overlay.html. Some CSS attributes are also integrated directly into the HTML file; these position and style the button and the frame of the pop-up, as well as the "*closePopupButton*". The actual chat interface is integrated as an iframe, which allows it to behave like a separate website. The button has a high z-index so that it is always displayed in the foreground.

The chat interface, defined in interface.html, can be understood as a separate website and can also be used as such by opening interface.html directly. This website is programmed more dynamically than the overlay and also uses two external resources as well as a configuration file (config.js) and a file for the logic of the interactive chat window (chat.js).

The external resources are Alpine.js, a small JavaScript framework for dynamic websites and tailwind.css, a CSS framework for simple class-based styling. These are not integrated via a package manager such as Node, but are stored locally in the package as a minified version. Alpine.js is used for the dynamic changes to the interface and also provides the basis for the logic that sends the chat messages to the API. Tailwind is the only stylesheet on this website.

The chat.js file contains all the logic for managing the chat history, for all animations and changes to the interface as well as for accessing the API or the proxy server. An Alpine.js app is first created here, which you can imagine to be a bit like an OOP class. It has methods and attributes. Alpine.js can be used to bind attributes of the app to inputs in HTML, which then automatically passes information in both directions. The input field *textarea* is linked to the input variable of the Alpine app by x-model="input". If the message is sent using the "*sendMessage*" button or ENTER, the input is appended to the previous messages and sent to the proxy server. Using SHIFT+ENTER, multi-line prompts can be entered and formatted as usual in chat applications. The reply is also added to the previous messages, the input is emptied and the chat window scrolls to the end of the conversation. The query headers allow a query to the proxy server without CORS.

The interface itself can be divided into two parts in terms of appearance. The chat history in the upper part consists of the logo and avatar, as well as the scrollable message container, while the lower part consists of the input area and the buttons for sending and deleting the conversation. The message container can display any number of messages and can be scrolled to view the entire history. Messages from the LLM are displayed on the left, messages from the user on the right. New messages trigger a function that automatically scrolls to the end of the conversation.

In addition to general settings, Config.js also lists the system prompt that defines the identity of the chatbot. An example is given. Changes to this system prompt change the identity and response behaviour of the chatbot. Prompt engineering is useful here. The system prompt should not be too long. The prompt length of all prompts has a strong influence on the costs of the application.

DigiKompASS is funded by the Foundation for Innovation in University Teaching.
Stiftung Innovation in der Hochschullehre

3.2 The function

The new function of h5p_adaptor.js binds the chat extension to the existing SCORM package. It is the only change to the existing code of the SCORM application.

The function is initially created as *setInterval*. This is a standard JavaScript function for a loop that is executed with defined delays. In this case, this is necessary and useful as the time at which all components were created by the SCORM package is not known. It is therefore necessary to wait until the interface of the escape room has been completely built up before adding new elements. The timeout here is one second. If the required element h5p-container is found, the overlay is loaded via the fetch function. A new div element is then created, the overlay inserted into it and added to the h5p-container as a new child. In addition, an *EventListener* is added to both the *openPopupButton* and the *closePopupButton*, which enables the popup to be opened and closed.

3.3 Proxy server

The proxy server forms an intermediate step between the plugin and the OpenAI API. It is designed for the use of Cloudflare Workers. The structure follows the templates for Cloudflare Workers, the code is only in one file, src/index.js. If the code needs to be recreated, the OpenAI library must also be imported. This library is already imported in the code provided. The Index.js of the proxy server definition receives the request from the chat interface, forwards it, enriched with the API key, which is defined in Cloudflare as an environment variable, to the OpenAI API, receives a response there and returns it to the chat interface. The header definitions prevent CORS errors.

DigiKompASS is funded by the Foundation for Innovation in University Teaching.

4 Instructions for use

1. Export your Lumi project with the Lumi Desktop application as a SCORM package (Lumi Web Export defective as of 02/2025)

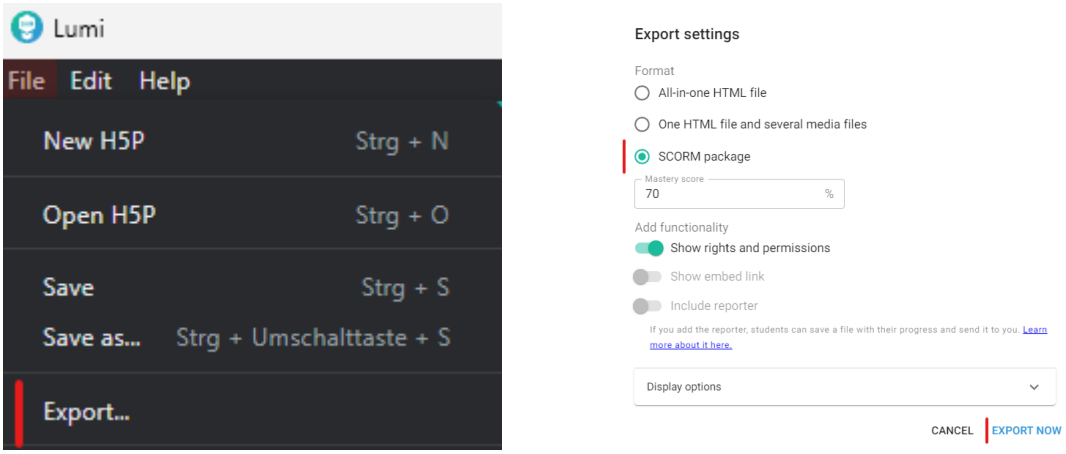

Figure 4 - Export settings

2. Unpack the archive into a folder
3. Download the code of the chat plugin from this Git repository: https://gitlab.uni-koblenz.de/egruen/gameloap (git clone or direct download)

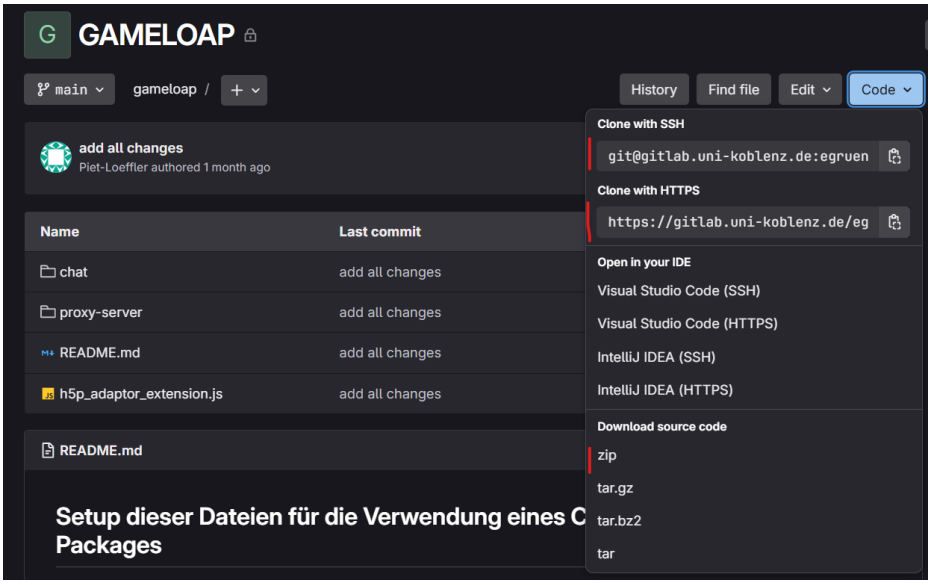

Figure 5 - Chat plugins from the Git repository

4. Read the Readme.md file completely
5. Open the file h5p_adaptor.js in your unpacked archive of the Escape Room with any text editor (NO MS Word, notepad++ or vscode recommended)
6. Open the file h5p_adaptor_extension.js from the downloaded files from GitLab with any text editor (see above)

DigiKompASS is funded by the Foundation for Innovation in University Teaching.
Stiftung Innovation in der Hochschullehre

7. Copy all contents of the ... extension.js file to a suitable location in the h5p_adaptor.js file (outside of existing functions, after the end() function is recommended, see image)

```
function end() {
    scorm.quit();
}

const intervalId = setInterval(function() {
    var elements = document.getElementsByClassName("h5p-container");
    if (elements.length > 0) {
        element = elements[0]
        fetch('./chat/overlay.html')
            .then(response => response.text())
            .then(data => {
                const newElement = document.createElement('div');
                newElement.innerHTML = data;
                element.appendChild(newElement);
                popup = document.getElementById("openPopupButton")
                popup.addEventListener("click", function() {
                    document.getElementById('popup').classList.toggle('hidden');
                })
                closePopup = document.getElementById("closePopupButton")

                closePopup.addEventListener("click", function() {
                    document.getElementById('popup').classList.toggle('hidden');
                })
            })
            .catch(error => console.error('Error loading HTML file:', error));
        clearInterval(intervalId);
    }
}, 1000);

window.onload = function () {
    init();
};
```

Figure 6 - ... extension.js code

8. Copy the /chat folder from the Git code into the Escape Room folder, it should now be "next to" the /images folder

chat	18.12.2024 15:55	Dateiordner	
images	18.12.2024 15:29	Dateiordner	
adlcp_rootv1p2.xsd	08.11.2024 07:34	XSD-Datei	5 KB
h5p-adaptor.js	20.12.2024 10:19	JavaScript-Quelld...	3 KB
ims_xml.xsd	08.11.2024 07:34	XSD-Datei	2 KB
imscp_rootv1p1p2.xsd	08.11.2024 07:34	XSD-Datei	15 KB
imsmanifest.xml	08.11.2024 11:23	XML-Datei	3 KB
imsmd_rootv1p2p1.xsd	08.11.2024 07:34	XSD-Datei	23 KB
index.html	08.11.2024 15:41	Chrome HTML Do...	1.846 KB
metadata.xml	08.11.2024 11:23	XML-Datei	5 KB
SCORM_API_wrapper.js	08.11.2024 07:34	JavaScript-Quelld...	30 KB

Figure 7 - Folder structure of the files

9. There is a config.js file in this copied /chat folder, customise it as you wish. You can also adjust the images in /chat/images if necessary. Always retain all file names when exchanging. The adjustments change the labelling of buttons, for example, or the behaviour and identity of the LLM.

DigiKompASS is funded by the Foundation for Innovation in University Teaching.

10. Once you have completed all the copying and customisation steps, package the Escape Room folder as a ZIP file and upload it to Moodle/Olat.

If no proxy server exists, you must create one. The instructions for this are here: https://developers.cloudflare.com/workers/get-started/guide/

The code for the proxy server is located in the /Proxyserver folder and can be uploaded to Cloudflare using npx wrangler deploy. No free account is required for this. The new proxy server must then be stored as a URL in config.js. The worker must also receive the API key as an environment variable in Cloudflare.

5 Notes

The OpenAI API costs money. The account that is currently behind it is assigned to Elisabeth Grün. Excessive use can lead to high costs.

Theoretically, it is possible to exploit and misuse this proxy server. 100,000 requests per day are free of charge.

This plugin has basically been developed as a prototype for research purposes. A productive long-term use requires investigation of security gaps, errors and probably requires further development of the code.

If you have any questions, please contact the person responsible for this project.

6 Problems

A number of problems arose during development:

1. CORS
    a. CORS or cross origin resource sharing is a security feature of Internet traffic
    b. CORS prevents requests from locally running systems to APIs, for example
    c. Bypass requires a proxy server in this case
    d. Proxy server must not be protected by CORS→ Unsafe!
2. SCORM
    a. SCORM is a format that basically describes ZIP files of HTML pages (+ logic for evaluation)
    b. SCORM packages are not well suited for customisation
    c. SCORM may behave differently locally than on OLAT, especially with regard to CORS
    d. API keys must never be stored in a SCORM package

DigiKompASS is funded by the Foundation for Innovation in University Teaching.

3. LUMI
    a. Lumi Web does not allow export to SCORM
    b. Lumi Web h5p files cannot be opened in Lumi Desktop (bug report was created by me)
    c. Changes to the escape room require re-embedding the chat
    d. There are h5p to SCORM formatted online, but these export parts of the SCORM code slightly differently than Lumi, which may require manual adjustments to the chat extension if these tools are used
4. Cloudflare
    a. The proxy server is not well secured
    b. The setup is very complex
    c. An AI gateway is required
    d. An account is required
    e. The proxy server is public, it could be scraped
    f. Technically, this part is somewhat more complex to recreate if it no longer exists.
5. Local development
    a. CORS and the OpenAI API behave differently depending on the context from which the call is made
    b. HTML imports as well (relative/absolute paths)
6. API key
    a. API keys must never be public
    b. Server is required, but SCORM does not provide one
7. LLM selection
    a. Initially, we considered hosting an LLM ourselves
    b. This would have required data centre resources
    c. The setup was complex
    d. CORS would have been a problem there too
    e. LLM data centre would have made VPN necessary

DigiKompASS is funded by the Foundation for Innovation in University Teaching.

8. Display

    a. The escape rooms may have problems displaying on unconventionally sized screens in full screen

    b. As the overlay is a pure overlay and is not anchored to elements of the Escape Room, it is possible that elements of the chat pop-up or the button may cover operating elements or content, which cannot be prevented. If this causes problems in operation, the browser window can be distorted to reveal elements.

DigiKompASS is funded by the Foundation for Innovation in University Teaching.