BOT QUIZZ GENERATOR

REPORT OF THE PROJECT

ESTEBAN VINCENT

DESTINED TO MRS. FATIMA MOQRAN

Table des matières

Introduction

During my 8th semester at Efrei Paris, I had the chance to attend an introductory course on Machine Learning and its different applications.

As part of this course, and with the aim of putting the different skills studied into application, we were asked to carry out a project with a free subject. I decided to create a Discord BOT to manage different quizzes, initially in the form of multiple choice questions and free questions.

I will come back in this report on the realization of the project, by first presenting ChatGPT and how I used it in the application, before talking about the realization of the project and presenting the different functionalities.

I – Use of the ChatGPT API

I – 1. What is ChatGPT?

ChatGPT is an artificial intelligence chatbot developed by OpenAI, designed to simulate conversation with human users. It’s based on OpenAI’s Generative Pre-trained Transformer (GPT) models, which are a type of large language model. Here are some key aspects of ChatGPT:

1. **Natural Language Processing (NLP):** ChatGPT uses advanced NLP techniques to understand and generate human-like text based on the input it receives.

2. **Training Data:** The model is trained on a diverse range of internet text, but it does not know specifics about which documents were part of its training set, nor does it have access to any personal data unless it has been shared with it during the conversation.

3. **Applications:** ChatGPT can be used for various applications including customer support, content creation, tutoring, language translation, and more. It’s designed to assist with answering questions, providing recommendations, and generating text based on prompts given by users.

4. **Interaction Style:** The interaction with ChatGPT is conversational, aiming to be context-aware and able to handle multi-turn dialogue, allowing it to maintain context over the course of a conversation.

5. **Versions:** There have been several iterations of ChatGPT, each improving on the previous in terms of capability, accuracy, and safety. GPT-3 and the subsequent GPT-4 models are among the most advanced, providing more coherent and contextually relevant responses.

6. **Limitations:** Despite its advanced capabilities, ChatGPT has limitations. It can sometimes generate incorrect or nonsensical answers, be sensitive to the phrasing of the

input, and may not have up-to-date information if events have occurred after its last training data update.

Overall, ChatGPT represents a significant advancement in AI and NLP, making human-computer interactions more seamless and intuitive.

I – 2. How does ChatGPT works?

ChatGPT works by leveraging a sophisticated machine learning model called a Transformer, which is specifically designed for handling natural language processing tasks. Here's a detailed look at how it functions:

1. Model Architecture:

- **Transformers:** The core of ChatGPT is based on the Transformer architecture. This architecture uses self-attention mechanisms to process input data efficiently, allowing the model to consider the context of a word based on surrounding words in a sentence or even in larger text segments.

- **Layers:** The model consists of multiple layers (e.g., GPT-3 has 175 billion parameters and 96 layers). Each layer refines the processing and understanding of the input text.

2. Training:

- **Pre-training:** During pre-training, the model learns to predict the next word in a sentence by being exposed to a vast corpus of text from the internet. This phase involves unsupervised learning, where the model learns language patterns, grammar, facts about the world, and some reasoning abilities.

- **Fine-tuning:** After pre-training, the model undergoes fine-tuning with supervised learning. It is trained on a narrower dataset with human reviewers following specific guidelines. This step helps the model learn to produce more accurate and useful responses based on given prompts.

3. Input Processing:

- **Tokenization:** When a user inputs a query, the text is tokenized. Tokenization breaks down the input text into smaller units called tokens, which could be words, subwords, or even characters. These tokens are converted into numerical format that the model can process.

- **Context Handling:** The model uses the tokens along with the context provided by previous interactions to generate a coherent response. It can maintain the context over multiple turns of conversation, allowing for more natural and relevant interactions.

4. Response Generation:

- **Decoding:** The model generates a response through a process called decoding. It starts with the given input tokens and generates subsequent tokens one by one, predicting the most likely next token at each step until it reaches the end of the response.

- **Beam Search/Top-k Sampling:** Techniques like beam search or top-k sampling may be used to improve the quality of the generated text. These methods help in balancing between generating the most likely response and ensuring diversity in the output.

5. Post-processing:

- **Detokenization:** The generated tokens are converted back into human-readable text through detokenization.

- **Filtering and Refinement:** Some systems may apply additional filtering or refinement steps to ensure the output is appropriate, coherent, and free from offensive content.

6. User Interaction:

- **Conversational Flow:** The model's ability to remember the context of the conversation enables it to provide relevant answers and engage in multi-turn dialogue, making interactions more natural and human-like.

- **Feedback Loop:** User feedback can be used to further fine-tune and improve the model over time, enhancing its accuracy and reliability.

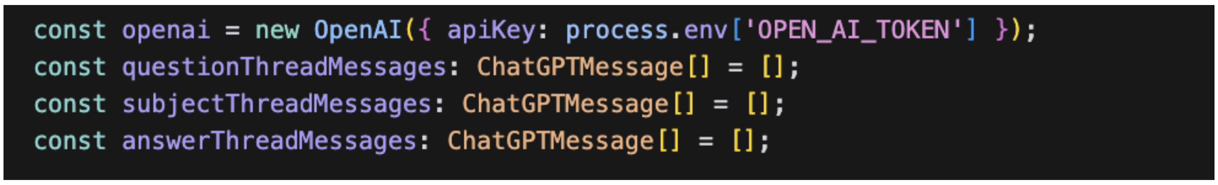
Overall, ChatGPT's operation involves a combination of sophisticated neural network techniques, vast amounts of training data, and advanced processing algorithms to understand and generate human-like text based on user input.

I – 3. Use of the ChatGPT API

I used the ChatGPT API with the official OpenAPI NPM package, which provides a whole range of options for using ChatGPT. In this section, I will go over the code that allowed me to use it.

First, I will initialize the various variables I will need, namely:

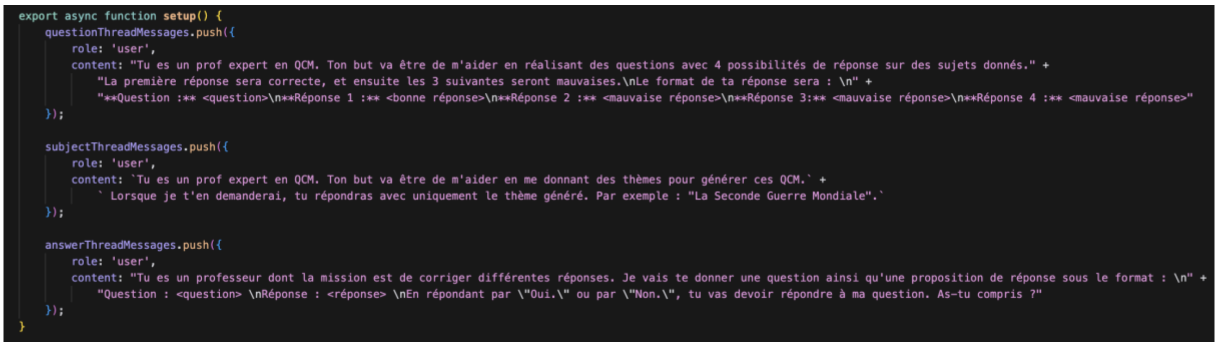
- A variable that will be an instance of OpenAI, on which we will make all our requests.
- Three variables that can store different conversations. We need 3 different chat spaces: one for generating questions, one for generating topics, and a last one for verifying answers (in case the quiz is not in the form of multiple-choice questions).



```
const openai = new OpenAI({ apiKey: process.env['OPEN_AI_TOKEN'] });
const questionThreadMessages: ChatGPTMessage[] = [];
const subjectThreadMessages: ChatGPTMessage[] = [];
const answerThreadMessages: ChatGPTMessage[] = [];
```

Code to initialize variables

Secondly, it is important to initialize the different conversations. To do this, we will add the initial prompts to the arrays containing each conversation. A “prompt” is an instruction that we give to ChatGPT to ask for information. To achieve the best possible results, and results that we can use later, we need to be as precise as possible about the content of the return message. Here is the code that allowed me to initialize the three different conversations.



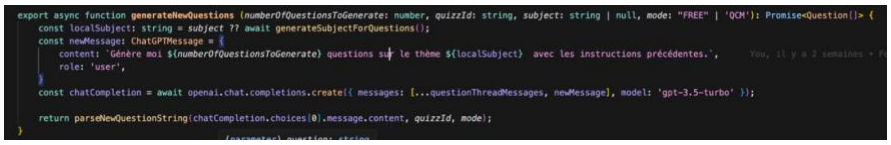
```
export async function setup() {
    questionThreadMessages.push({
        role: 'user',
        content: "Tu es un prof expert en QCM. Ton but va être de m'aider en réalisant des questions avec 4 possibilités de réponse sur des sujets donnés." +
            "La première réponse sera correcte, et ensuite les 3 suivantes seront mauvaises.\nLe format de ta réponse sera : \n" +
            "**Question :** <question>\n**Réponse 1 :** <bonne réponse>\n**Réponse 2 :** <mauvaise réponse>\n**Réponse 3:** <mauvaise réponse>\n**Réponse 4 :** <mauvaise réponse>"
    });

    subjectThreadMessages.push({
        role: 'user',
        content: `Tu es un prof expert en QCM. Ton but va être de m'aider en me donnant des thèmes pour générer ces QCM.` +
            ` Lorsque je t'en demanderai, tu répondras avec uniquement le thème généré. Par exemple : "La Seconde Guerre Mondiale".`
    });

    answerThreadMessages.push({
        role: 'user',
        content: "Tu es un professeur dont la mission est de corriger différentes réponses. Je vais te donner une question ainsi qu'une proposition de réponse sous le format : \n" +
            "Question : <question> \nRéponse : <réponse> \nEn répondant par \"Oui.\" ou par \"Non.\", tu vas devoir répondre à ma question. As-tu compris ?"
    });
}
```

Initialization of conversations

The main feature of the project is quiz generation. Therefore, we will first look at the code that allowed us to achieve this. Next, we will see how we were able to generate topics in case the user decides not to define topics for the quiz. We will then see how we were able to use the textual content of ChatGPT's response to generate a complete question, including all the steps. Finally, we will show how we were able to verify that the user's answers are correct.

I – 3. A. Generation of Questions

To generate the different questions, we used the previous variables, namely the instantiation of OpenAI and the messages related to question generation.
First, we need to generate the topic of the quiz to be generated. There are two possible options for this. The first is if the quiz topic is already generated, we will simply use it. The second is to generate it using the OpenAI API. We will return to this second method later.
Next, we had to generate the final object containing the new message to be generated and concatenate it with the previous messages. I will then send all these messages to the OpenAI API, using the GPT-3.5 Turbo model. Following various tests, I concluded that this was the model with which I obtained the best results.
Finally, we will need to analyze ChatGPT's response to convert a textual response into a table of objects usable by my BOT. I will detail the algorithm I implemented for this later.



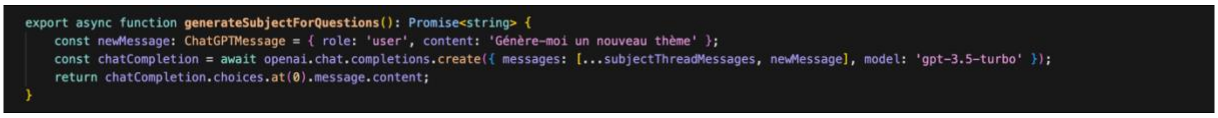
```
export async function generateNewQuestions (numberOfQuestionsToGenerate: number, quizzId: string, subject: string | null, mode: "FREE" | 'QCM'): Promise<Question[]> {
    const localSubject: string = subject ?? await generateSubjectForQuestions();
    const newMessage: ChatGPTMessage = {
        content: `Génère moi ${numberOfQuestionsToGenerate} questions sur le thème ${localSubject} avec les instructions précédentes.`,
        role: 'user',
    }
    const chatCompletion = await openai.chat.completions.create({ messages: [...questionThreadMessages, newMessage], model: 'gpt-3.5-turbo' });

    return parseNewQuestionString(chatCompletion.choices[0].message.content, quizzId, mode);
}
```

Code for generating questions

I – 3. B. Topic generation

During the generation of quizzes, we found that our BOT could automatically generate quiz topics if the user prefers to leave this task to our BOT. In this section, we will present how we achieved this.

To do this, we will need to use the initialization of previously created variables, particularly the instantiation of the OpenAI class and the one containing the messages related to the topic generation chat.
The method that allows us to generate a new theme is straightforward: we first need to ask ChatGPT to imagine a new theme, and ensure the prompt only contains the generated theme. Following this, whenever we need a new theme, we request it and simply return the generated text, as it will only consist of the generated theme.

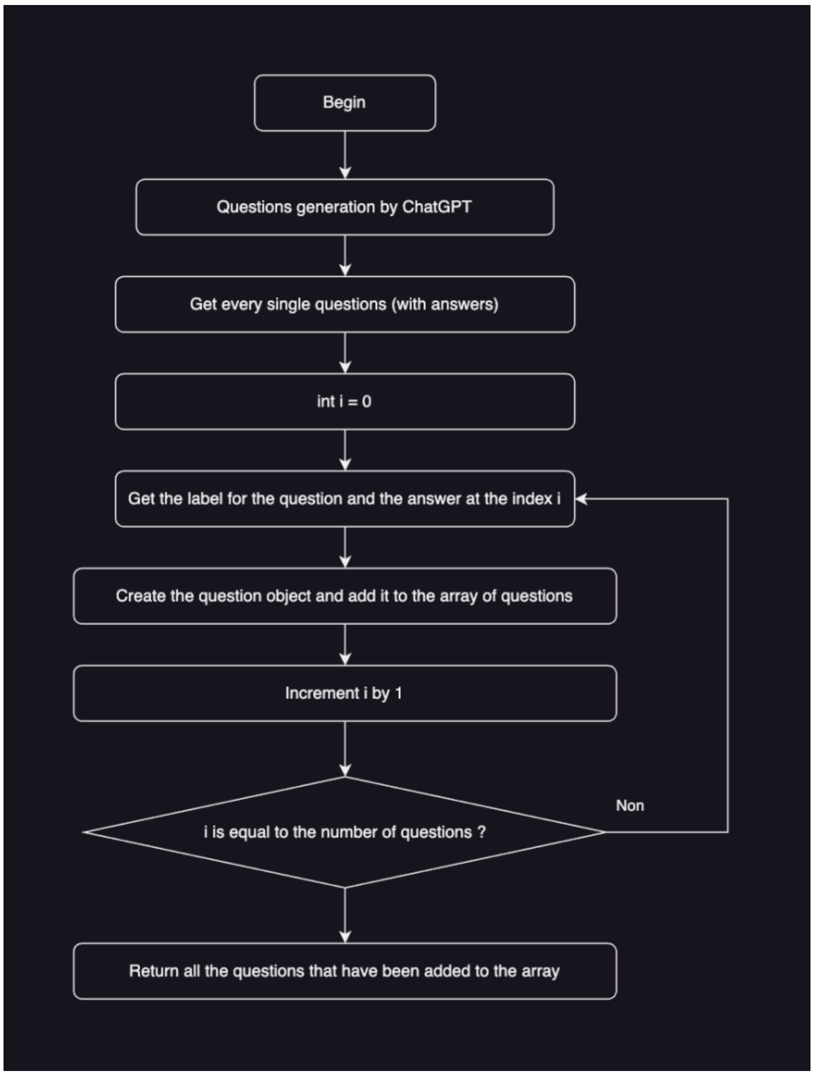


```
export async function generateSubjectForQuestions(): Promise<string> {
    const newMessage: ChatGPTMessage = { role: 'user', content: 'Génère-moi un nouveau thème' };
    const chatCompletion = await openai.chat.completions.create({ messages: [...subjectThreadMessages, newMessage], model: 'gpt-3.5-turbo' });
    return chatCompletion.choices.at(0).message.content;
}
```

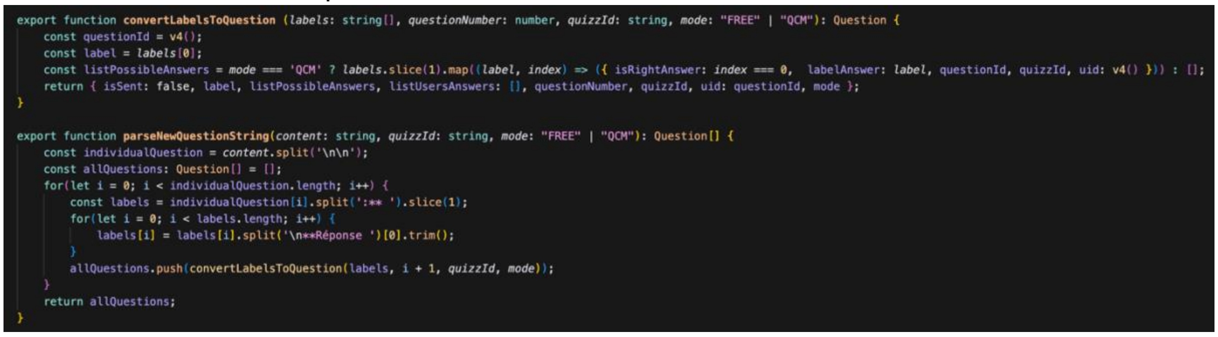
Code for generating topics

I – 3. C. Utilization of ChatGPT responses

As explained previously; after receiving the different questions from ChatGPT, we obtain a response in the form of character strings. We will therefore have to write an algorithm to exploit this response and form the object that can be used in our project. So here I will present the algorithm to achieve this. Here is the Workflow Diagram of how I've achieved this (with question generation).


Begin
Questions generation by ChatGPT
Get every single questions (with answers)
int i = 0
Get the label for the question and the answer at the index i
Create the question object and add it to the array of questions
Increment i by 1
i is equal to the number of questions ?
Non
Return all the questions that have been added to the array

Here is the code of this part :



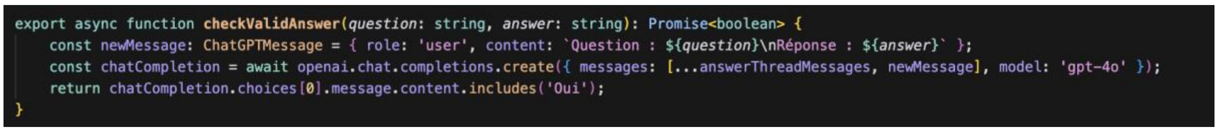
```
export function convertLabelsToQuestion (labels: string[], questionNumber: number, quizzId: string, mode: "FREE" | "QCM"): Question {
    const questionId = v4();
    const label = labels[0];
    const listPossibleAnswers = mode === 'QCM' ? labels.slice(1).map((label, index) => ({ isRightAnswer: index === 0,  labelAnswer: label, questionId, quizzId, uid: v4() })) : [];
    return { isSent: false, label, listPossibleAnswers, listUsersAnswers: [], questionNumber, quizzId, uid: questionId, mode };
}

export function parseNewQuestionString(content: string, quizzId: string, mode: "FREE" | "QCM"): Question[] {
    const individualQuestion = content.split('\n\n');
    const allQuestions: Question[] = [];
    for(let i = 0; i < individualQuestion.length; i++) {
        const labels = individualQuestion[i].split(':** ').slice(1);
        for(let i = 0; i < labels.length; i++) {
            labels[i] = labels[i].split('\n**Réponse ')[0].trim();
        }
        allQuestions.push(convertLabelsToQuestion(labels, i + 1, quizzId, mode));
    }
    return allQuestions;
}
```

Code for utilizing and parsing ChatGPT responses

I – 3. D. Response verification

Finally, the last step in which the use of the OpexnAI API is necessary is the verification of the answers, for open questions. To do this, we will insert the question as well as the answer in the messages, asking them to only respond with "Yes." » or by "No. ". So, we can very easily retrieve whether the answer is correct or not: we just have to look if the content of the ChatGPT message contains the word "Yes".

Concerning the choice of model, I chose to go with the GPT 4o model after various tests which allowed me to conclude that for this task, it was the most suitable model.



```
export async function checkValidAnswer(question: string, answer: string): Promise<boolean> {
    const newMessage: ChatGPTMessage = { role: 'user', content: `Question : ${question}\nRéponse : ${answer}` };
    const chatCompletion = await openai.chat.completions.create({ messages: [...answerThreadMessages, newMessage], model: 'gpt-4o' });
    return chatCompletion.choices[0].message.content.includes('Oui');
}
```

Code for verifying responses

Thus, thanks to these different and very simple steps, we were able to manage all the steps allowing the generation of multiple-choice, free-response quizzes, the verification of the answers, and the generation of themes.

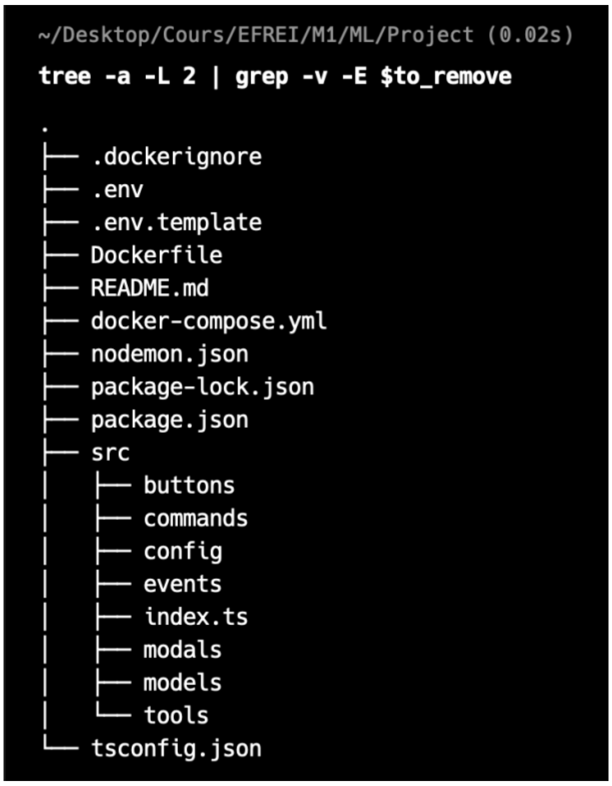
II - Project realization

II – 1. Project description

The project consists of the creation of a Discord BOT, working with the OpenAI API for generating questions. The BOT has two modes: the mode with multiple choice questions, and the mode with open questions.
I will come back later to the different methods that allowed me to carry out this project.

II – 2. Project structure

This section will be shorter than the part presenting my use of the OpenAI API. It will simply outline the basic structure of the project without delving into too many technical details.


```
~/Desktop/Cours/EFREI/M1/ML/Project (0.02s)
tree -a -L 2 | grep -v -E $to_remove

.
├── .dockerignore
├── .env
├── .env.template
├── Dockerfile
├── README.md
├── docker-compose.yml
├── nodemon.json
├── package-lock.json
├── package.json
├── src
│   ├── buttons
│   ├── commands
│   ├── config
│   ├── events
│   ├── index.ts
│   ├── modals
│   ├── models
│   └── tools
└── tsconfig.json
```
Screenshot of the project directory structure

As we can see with the screenshot of the project tree (made using the tree command), we have different files at the root. These are project configuration files, and there are notably.

- Files used for the Dockerization of the project: .dockerignore, Dockerfile and docker-compose.yml
- The files for the environment variables: .env (which will store the values and which will have to be ignored by Git so as not to share confidential information) and a .env.template, which will contain the different names for the environment variables. environment associated with the project.
- The README, which will contain a very quick description of the project as well as the different steps to launch the project. To find out more about launching the project, you can refer to this part.
- Dependency files: package.json and package-lock.json. These two files will contain all the information about the different dependencies of the project. The dependencies will be stored in a node_modules folder, located at the root, but which I decided to remove from the tree command display to maintain a cleaner display.
- The TypeScript language configuration file: tsconfig.json. It is in this file that we will define the different configurations, compiler rigidity levels, entrypoint, build folder, and other information relating to TypeScript.
- The Nodemon configuration file: nodemon.json. Nodemon is a monitoring tool allowing you to restart a JavaScript project when modifying a project file. It is a watcher widely used in JavaScript (or TypeScript) project development.

In addition, we also have a folder: the src folder, which will contain different folders and the index.ts file, which is the entry point of the program. The src folder contains the subfolders:

- The configuration folder: config/. This folder will allow us to store the different configurations for our application. Here we have placed the configuration file for our database.
- The models/ folder. This folder contains everything related to the database: connection, definition of different tables, data control, etc.
- The DiscordJS buttons and the different interactions are managed from the buttons/ folder.
- The different Slash Commands available thanks to the BOT are in the commands/ folder.
- The different Discord events are in the events/ folder. This can include starting the BOT or creating user interactions for example.
- The different submissions to Discord modal windows are managed in the modals/ folder. Right now, it's only when the user answers an open-ended question, but there may be more use cases for this part in future releases.
- The tools/ folder contains all the tools necessary for the application to run smoothly. This includes for example features related to the ChatGPT API, date functions, handler functions, etc.

II – 3. List of the features

A – Generate quiz with QCM

The main feature, which is the reason the BOT was created, is the generation of quizzes with multiple-choice questions (MCQs). This involves generating questions using the OpenAI API and then sending them out when the quiz date arrives.

B – Generate quiz with open questions

An equally important feature, added later, is the generation of quizzes with open-ended questions. For this, we will use the ChatGPT AI to generate the questions, store them in our database, and send them out when the quiz time arrives. After that, users will type their responses in a modal, and a second part of the program will verify the answers.

C – Show the ranking

It can be very interesting to look at the ranking. For this, the application has two different rankings:

- The first is the global ranking, which has existed since the BOT was put into production and therefore includes all quizzes. To view this, you can use the Slash Command /show_ranking.
- The second is the ranking by quiz, which means the ranking for a particular quiz. To find this, you need to go to the channel corresponding to the desired quiz, and you will find it in the first message sent by the BOT.

D – Delete Quiz

Another important feature is the deletion of quizzes. Indeed, this allows the cancellation of quizzes if they were created incorrectly, the deletion of quizzes after their launch to maintain order, to free up space in the database, among many other advantages.

II – 4. Installing & launching the project

II – 4. A. From Scratch

If you want to run your own Quizz Generator BOT, you can do so. I added a README file at the root of the project that sums up the way you can achieve this. You must follow the different instructions, and this will normally work well. If you have any question, you can text me on the 07 80 44 45 79 for quick answer.

II – 4. B. With an already set up project

The process to create your own Quiz Generator BOT is hard. This is why I have hosted a BOT, so you can easily use it. To use it, you have just to follow this link and join the Discord's server that I created for the project. After that, you can enjoy the BOT as easily as that.

Conclusion

Thus, in this project, we were able to see what AI is, specifically ChatGPT, how it was built, and how it functions. We also explored how we can use its API and demonstrated this through a specific project example: a Discord BOT managing different types of quizzes on various subjects, and the generation of quiz topics.

We could envision more features for the Discord BOT to continue the theme of learning and culture. For example, we could include functionalities related to learning interesting anecdotes and lessons on various topics, sending a certain number of anecdotes automatically at set intervals, personalizing the anecdotes sent, verifying sentences, among many others.