

Capstone Project Phase B

Software Engineering Department

Braude College

Project number: 24-1-D-11



GPTalk – Learning languages, enhanced for the modern age

**Supervisor**  
Alex Keselman

**Students**  
Oneill Panker [oneill.panker@e.braude.ac.il](mailto:oneill.panker@e.braude.ac.il)  
 Roman Gury [roman.gury@e.braude.ac.il](mailto:roman.gury@e.braude.ac.il)

**GitHub Repository link**: [**GPTalk**](https://github.com/Oneill19/GPTalk)

Table of Contents

[Abstract 4](#_Toc177221383)

[Introduction 4](#_Toc177221384)

[Product Description and Structure 4](#_Toc177221385)

[Overview 4](#_Toc177221386)

[Key Features 4](#_Toc177221387)

[Interface Design 5](#_Toc177221388)

[Learning Material and Content Generation 5](#_Toc177221389)

[Our Users 6](#_Toc177221390)

[System Architecture and Diagrams 7](#_Toc177221391)

[Architecture 7](#_Toc177221392)

[Deployment Diagram 7](#_Toc177221393)

[Package Diagram 8](#_Toc177221394)

[Activity Diagram 9](#_Toc177221395)

[Use Case Diagram 10](#_Toc177221396)

[Development Process 11](#_Toc177221397)

[Initial Research 11](#_Toc177221398)

[Technology Stack 11](#_Toc177221399)

[Backend 12](#_Toc177221400)

[Frontend 13](#_Toc177221401)

[Integration 13](#_Toc177221402)

[Consultations with potential customers 13](#_Toc177221403)

[Challenges and Solutions 14](#_Toc177221404)

[Results and Conclusions 15](#_Toc177221405)

[Lessons Learned 16](#_Toc177221406)

[Verification Testing 17](#_Toc177221407)

[Frontend Testing 17](#_Toc177221408)

[Backend testing 21](#_Toc177221409)

[Unit Testing 22](#_Toc177221410)

[API Testing 22](#_Toc177221411)

[User Guide 24](#_Toc177221412)

[Login and Registration 24](#_Toc177221413)

[Home Screen 26](#_Toc177221414)

[Learn 27](#_Toc177221415)

[Fill in the Blank 28](#_Toc177221416)

[Translate the Word 29](#_Toc177221417)

[Translate the Sentence 29](#_Toc177221418)

[Complete The Conversation 30](#_Toc177221419)

[Match The Words 31](#_Toc177221420)

[Reorder Sentence 31](#_Toc177221421)

[Match The Category 33](#_Toc177221422)

[Summarize the Paragraph 35](#_Toc177221423)

[Choose the Tense 35](#_Toc177221424)

[Chat With Me 36](#_Toc177221425)

[Leaderboard 39](#_Toc177221426)

[My Profile 40](#_Toc177221427)

[About 43](#_Toc177221428)

[Maintenance Guide 44](#_Toc177221429)

[References 45](#_Toc177221430)

# Abstract

GPTalk is a language learning web application, developed with the aim of making the language acquisition process more varied and robust, while providing an enjoyable and user-friendly experience. The application provides a gamified approach to the process, featuring levels, leaderboards, and achievements, and allows the user to improve their skills by partaking in a variety of randomly generated exercises, and participating in conversations with a chat-bot. Both types of content are generated with the help of OpenAI's GPT-4o large language model, ensuring that the learning material provided to the user is diverse, educational, and engaging.

GPTalk takes the idea of a gamified educational application to the next level by taking advantage of recent developments in machine learning and large language models and combining them with a user-friendly interface and levels of difficulty suitable for each student's skill level.

# Introduction

Our project utilizes OpenAI’s latest large language model, GPT-4o, to generate educational exercises designed to improve the skills of students who are learning new languages, in addition to serving as a chatbot that allows the students to practice their knowledge by chatting with it.

GPTalk focuses on the learning method of gamification to teach the students:  
The web application allows the users to track their progress by saving their statistics such as mistake-free lessons, consecutive usage of the app, and their worldwide ranking. Additionally, it saves their high scores, knowledge level in each language, and awards the students with achievements for getting to various milestones.

# Product Description and Structure

## Overview

GPTalk is a web application designed to help users learn languages by integrating gamification elements and combining them with OpenAI API’s state-of-the-art ability to generate content of great variety and high precision in multiple languages. The application provides an engaging, interactive platform for language learning, optimized for both computers and mobile devices. Its clear, simple, and user-friendly interface features large text and intuitive icons, ensuring an accessible experience for users of all ages and technical backgrounds.

## Key Features

1. **Learn**:
   * This feature allows users to engage in “lessons,” which are groups of various language learning exercises, available in multiple categories, each one providing a different objective aimed at improving the user’s knowledge of the language he is learning.
   * Exercises are dynamically generated using the OpenAI API, offering a level of variety and adaptability beyond traditional language learning applications.
   * Users earn experience points (EXP) based on their performance, which contributes to their overall progress and motivation.
2. **Chat with Me**:
   * This feature enables users to practice their newly acquired language skills in a conversational setting with an AI-powered chatbot, which dynamically interacts with the user based on their needs.
   * The chatbot simulates real-life conversations, helping users to apply what they’ve learned in practical, everyday scenarios.
3. **Progress tracking and Gamification**

* Users create an account to interact with the application, and their learning progress is tracked through detailed statistics, which display their total gained experience, recent results, and performance over time.
* The application features various achievements that users can unlock by consistently using the app and performing well in their lessons. These achievements serve as milestones and motivate users to continue learning.
* The application includes a leaderboard feature, allowing users to compare their scores with others, fostering a sense of community and friendly competition.

## Interface Design

The design of GPTalk is centered around clarity and ease of use:

* The interface is designed to be intuitive, with large, easily recognizable icons and text that guide the user through the application’s features.
* Navigation is straightforward, ensuring that users can quickly access the "Learn" and "Chat with Me" features without unnecessary steps.
* The application is responsive, providing a seamless experience across different devices, whether on a desktop, tablet, or smartphone.

## Learning Material and Content Generation

GPTalk sets itself apart from other language learning applications by utilizing the OpenAI API for content generation. The power of OpenAI’s Large Language Models allows the app to create a diverse range of exercises that are tailored to the user's learning pace and style:

* The content is generated using OpenAI’s latest GPT-4o mini model, which provides very fast responses while maintaining the accuracy of the flagship GPT-4o model, but costing significantly less per API request.
* The requests sent to the API are dynamically generated based on the language the user is learning, the user’s progress in the chosen language, and the exercise’s category which is randomly chosen in advance.  
  All these parameters determine the difficulty of the generated material, the number of choices the user will be given, the length of the sentences he will need to complete or write, and more. To help with the diversity of the generated material, random topic keywords from our own list are added to the prompts sent to the API during the lesson generation. This allows the generated results to be less deterministic and more unpredictable.
* The AI-generated content ensures that users are always presented with fresh and challenging material, preventing the monotony often associated with repetitive learning exercises.

# Our Users

GPTalk is designed with a broad audience of language learners in mind, regardless of their current proficiency level. Whether you're a complete beginner just starting out, or an advanced learner looking to refine your grammar and vocabulary, GPTalk provides a flexible and supportive environment to aid in your language learning journey.

Our users come from various backgrounds and have different reasons for wanting to learn a new language. Some might be preparing for travel, others want to learn for personal enrichment, while a different group is pursuing professional development. GPTalk meets these diverse needs by offering content that is adaptable and accessible, making it an effective resource for all these users.

GPTalk is not intended to replace traditional language learning methods but rather to complement them. We recognize that mastering a language requires multiple resources and consistent practice. GPTalk serves as an engaging and motivating tool to help users stay on track and make steady progress. By incorporating GPTalk into their study routine, users can reinforce what they’ve learned elsewhere and gain additional practice in a fun, interactive format.

Our goal is to inspire users to continue their language learning journey beyond the app. We believe that learning a language is an ongoing process, and GPTalk is here to support and encourage users along the way. By providing accessible, high-quality learning experiences, we help users build a solid foundation and develop the confidence to pursue further language studies on their own terms.

# System Architecture and Diagrams

## Architecture

A diagram of a process

Description automatically generated

Figure 1 GPTalk architecture diagram

## Deployment Diagram

A diagram of a software application

Description automatically generated

Figure 2 Deployment Diagram

## Package Diagram

A diagram of a diagram

Description automatically generated

Figure 3 Package Diagram

## Activity Diagram

A screenshot of a computer screen

Description automatically generated

Figure 4 Activity Diagram - learning session

A screenshot of a computer

Description automatically generated

Figure 5 Activity Diagram - chat session

## Use Case Diagram

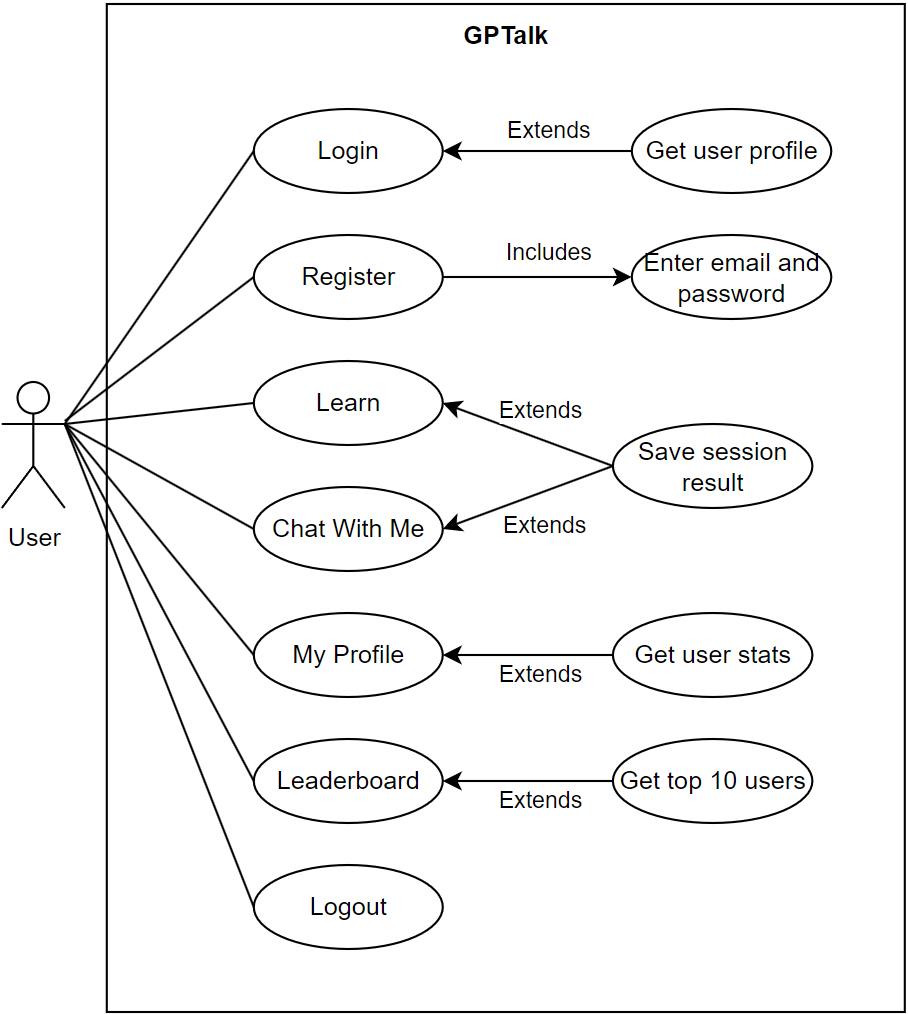


Figure 6 Use case diagram

# Development Process

## Initial Research

Before starting active development of the application, we conducted research to assess the capabilities and functionalities of OpenAI’s API, to better understand how it could be effectively utilized to generate engaging and educational language learning tasks. Various test prompts were given to ChatGPT which uses LLM’s akin to what our eventual app would utilize. The impressive results generated by ChatGPT showed us that this application could work and gave us the confidence to proceed with our idea.

To achieve a clearer idea of how GPTalk would look and function, we analyzed existing language learning applications, such as Duolingo and Babbel, to understand their strengths and limitations. We looked at feedback and analysis of the existing language learning solutions that utilize gamification to identify common problems and areas for improvement. The goal was to identify what our application could provide that the previous ones lacked.

## Technology Stack

Following our research, our focus shifted towards the architectural and technological considerations necessary to bring GPTalk to life. We selected a tech stack that would not only ensure high performance and scalability but also align with our vision of creating a seamless and interactive user experience.  
For the frontend, we chose Angular, thanks to our prior experience with it and the strengths it had over its alternatives, such as its component-based structure.  
For the backend, we chose Express.js, a Node.js web application framework, due to its lightweight nature and flexibility. It allows us to build a scalable and efficient API that seamlessly integrates with Angular. Express.js’s middleware capabilities simplify request handling and error management, ensuring smooth communication between the frontend and backend. This choice also supported our performance resposiveness needs, crucial for GPTalk’s interactive experience.  
To store and structure the application’s data, we used MongoDB. It stores the user’s personal data, the results of all the lessons he has done, and logs of his visits. Additionally, more general data is stored there, like the daily words.  
The user authentication code is also found in the backend – controller files contain functions that log the user into the app, add newly registered users, and retrieve the current user’s data.

Backend  
When a user is created, the passwords saved in the database are hashed on the backend side using the Bcrypt library for extra security. Additionally, a “registration complete” email is sent to the user with the help of the nodemailer library.  
  
The data we use to connect to our database and the API key, and all other secrets of our application is stored in environment variables in the .env file.

Another key aspect of the backend was the integration of the OpenAI API. To generate an exercise, a prompt message needs to be constructed in such a way that the API will consistently return a coherent response that can be converted to an exercise file. To maximize the accuracy of the responses, the prompt message for each exercise category is written as concisely and clearly as possible and requests the minimal amount of information needed to construct an exercise object.  
The connection to OpenAI is made in the lesson-generator controller file, where the exercise generation prompt is received from the frontend’s lesson-generator service and is sent to the API using the fast and inexpensive gpt-4o-mini model.   
To get the results we need, we used the Chat Completions API, which supports text inputs and can return text content in a JSON format, which was the most convenient option for our needs when creating the exercise data. In the request object that we sent to the API we specified the output type to be JSON, and the temperature, which controls the API’s response randomness, was set to 0.3, slightly more than the default value, to increase variety, but not too high so the results would still match the structure of the requests.  
Once a response has been acquired, it is sent back to the frontend and is processed from a raw JSON file into an Exercise type object.  
For the “Chat with me” feature, the connection to OpenAI is handled similarly but with additional logic for interaction. When a user initiates a chat, the input is sent to the GPT-4o model via the chat completion API. The model responds not only with a continuation of the conversation but also provides feedback and follow-up questions based on the user’s input. At the end of the chat session, the model automatically grades the conversation, offering users insights into their language proficiency. The response data, including feedback and grades, is processed into a structured format before being sent back to the frontend.

The API prompt quality was tested using OpenAI’s built in Playground tool, where various parameters and settings can be quickly tested and adjusted to get the best results.   
The backend’s general functionality was tested using Postman to validate API endpoints and ensure the reliability and performance of the application.

For unit-testing, Jest was used to ensure that the backend controllers functioned correctly.

## Frontend

On the frontend side, the PrimeNG library, together with the PrimeFlex extension were used to construct the user interface, offering a wide range of UI components and layout options that aligned with the application’s design goals. Additionally, SCSS was utilized for its extra features such as variables and nesting.

Toastr was integrated to manage notifications, providing real-time feedback to users, specifically on login and registration attempts.

Throughout the frontend development process, emphasis was placed on creating a simple and readable interface that would be easy to navigate for users of all ages and technical abilities. Therefore, from the earliest prototype of the user interface, it was decided that every main feature would be accessible from every screen via the sidebar, to reduce the amount of clicks the user would need to perform to access the feature of his choice. That was also the reason for the large typefaces and buttons across the application, and to make the options clearer, icons were added next to every menu item.  
To make the various exercises provided to the user more understandable, each one comes with a concise sentence describing that exercise’s goal, and more detailed instructions are provided at the bottom of the screen, to save the user the trouble of going to separate menus to learn how to perform the given exercise.  
Additionally, to help the user learn from his mistakes, the prompt’s translation appears when the exercise ends – for example in a “complete the conversation” exercise, the translated conversation appears.

For unit-testing, Jest was used to ensure that the frontend components functioned correctly and provided a seamless experience for the user.

## Integration

For version control, and an effective collaborative process using Git integration, we used GitHub.  
The frontend and backend are integrated to communicate seamlessly via RESTful API endpoints.

## Consultations with potential customers

Throughout the development process of GPTalk, we engaged in consultations with potential users to gather feedback, validate our assumptions, and refine our product roadmap. By soliciting input from language learners, we gained valuable insights into their needs, challenges, and preferences regarding language acquisition. This feedback guided our decision-making process and informed feature prioritization, ensuring that our application addresses real-world user requirements by offering a gamified, engaging, and personalized language learning experience.

# Challenges and Solutions

During the development process we encountered numerous challenges that we had to overcome to ensure our application meets its requirements:

* Constructing the prompts for OpenAI’s API proved to be a challenge: The initial plan was to construct a single detailed prompt for a single lesson by dynamically constructing a single string with all the exercises that we want to generate. This, however, proved to be an unreliable method, because the prompt's length caused the GPT model some confusion and led to inconsistent results.   
  We then decided to send the prompts for each exercise separately, to reduce the chances of the model making mistakes. The change helped, but the prompts were still causing inconsistent results: The problem was the fact that the prompts asked for specific and complex JSON object structures, that the model did not always understand.   
  For example, in a “fill in the blank” exercise we would ask the model to return an object with a sentence that has a missing word, and an array of words including that missing word, but the API often returned results that only partially matched the request. To address the issue, we simplified the prompt to request a complete sentence, and an array of random words. The API had no issues returning suitable results, and we added a custom function that constructed a proper “fill in the blank” exercise from the API’s response.  
  After we simplified all prompts and added functions to transform the results into proper exercises, the problem was solved and we got accurate and consistent generated material for our lesson generator.
* Another problem with OpenAI’s generated material was a lack of variety. For identical exercise prompts, exercises were being generated in a deterministic manner, with little to no difference from one another. To address the issue, we added a file with numerous topics that we added to the exercise prompts at the time of generation, that added randomness to the generated results and addressed the issue of variety.
* Our initial testing was done with the GPT-4o model, which is the most recent and advanced one at the time of writing. The model we were interested in however, was GPT-4o-mini, which was almost identical in its abilities, but significantly cheaper and therefore preferrable if the application was to be used extensively. However, the length of our initial generation prompt was longer than what 4o-mini could handle, so we had to use the regular 4o.   
  However, once we switched to per-exercise generation instead of generating whole lessons at once, our prompts became short enough for 4o-mini to handle, so we were able to take advantage of it.
* The structure of the Learn component was another issue – the initial implementation contained all exercise templates in a single file. This, however, proved to make the code too cumbersome, so a decision was made to split all exercise types into separate components to improve the code’s readability.

# Results and Conclusions

The development of GPTalk was guided by a clear set of goals: to create an engaging and user friendly language learning application that utilizes modern technology, offers diverse content, focuses on gamification, and provides a user-friendly interface. With the help of extensive research, strategic planning, and the use of a well-rounded technology stack that we had prior experience with, we successfully achieved these goals.

By incorporating gamification elements such as achievements, stats tracking, and leaderboards, we managed to create an app that would be able to motivate users to continue their language learning journey.   
Utilizing the OpenAI API for content generation allowed us to offer a wide range of exercises that adapt to the user's progress and needs. This variety in content kept users challenged and interested, preventing the monotony that can often occur in language learning apps.  
The decision to use Angular, PrimeNG, and PrimeFlex for frontend development ensured that GPTalk’s interface was both intuitive and responsive. By prioritizing clear navigation and accessibility, we created a platform that is easy to use for learners of all ages and technical abilities.

The development process was not without its challenges, but each obstacle provided an opportunity to refine our implementation and improve the final product. We refined the OpenAI API implementation to improve the diversity and accuracy of the generated content, and conducted testing using the OpenAI Playground to fine-tune the prompts and parameters used in the generation functions. We also switched to the GPT-4o-mini model mid-development, which allowed for fast content generation at a minimal cost.  
By utilizing Bcrypt in the password creation process and by generating JWT tokens on user authentication we improved the application’s security, making it safer from potential attackers.

Several key design decisions were made during the development process, each with a specific goal in mind: Angular’s component-based architecture was chosen for its ability to manage complex state and reusable components efficiently. The choice of MongoDB as the database was driven by its flexibility in handling diverse data types and its scalability. This allowed us to store user progress, achievements, and other dynamic data efficiently, while also being able to adapt to future changes in the data model.  
The use of PrimeNG and SCSS for styling and UI design enabled us to easily implement an interface that supported responsive design.

Not every goal we set for ourselves was met: Due to development time constraints, we were unable to implement some features that were part of our initial goals. Specifically, the inclusion of a dark mode and full mobile support were not realized in the current version. We recognize that these features are important for user comfort and accessibility, however we focused on the main features of the application.

In conclusion, the development of GPTalk was a complex but rewarding process that successfully met the project's initial objectives. There is always room for improvement in a web application of this kind, and we have several ideas for features that can be added in the future to improve it, but we believe we managed to create a feature-complete product for the limited development time we had to work with. By overcoming challenges and making informed design decisions, we created a language learning application that is both effective and enjoyable.

# Lessons Learned

The development process of GPTalk taught us numerous valuable lessons, and provided us with knowledge in various areas of full-stack development:

* It improved our understanding of developing full-stack applications - this project required us to integrate multiple technologies, from MongoDB, the OpenAI API and Express.js on the backend to Angular and PrimeNG on the frontend. Each component needed to work seamlessly together so the application would be able to provide the users with a cohesive and enjoyable experience. We learned the importance of designing a robust backend architecture that could efficiently handle data management, API requests, and user authentication. On the frontend, we gained a deeper appreciation for creating intuitive and aesthetically pleasing user interfaces.
* It showed us the importance of conducting relevant research. This phase was instrumental in shaping the direction and features of GPTalk. By thoroughly examining existing language learning platforms and understanding user needs, we were able to identify the areas our application could focus on. This helped us decide in advance the ways we could integrate gamification elements, and utilize the latest developments in AI for content generation. It also helped us address issues observed in existing applications, such as overly complex interfaces or a lack of content variety.
* It demonstrated the value of constant communication and discussions during every stage of development. From the initial planning stages to the final deployment, we had regular meetings that involved discussions and updates, ensuring that we were on the same page. This constant communication helped us quickly identify and address potential issues, share ideas, and make collective decisions that aligned with the project’s goals. The constant collaboration ensured that we both could take advantage of our strengths to improve the overall quality of the application. The consistent meetings and communication also ensured that we never fell behind schedule throught the developlemt process.
* It showed us the strengths but also the limitations of using LLMs for content generation. Integrating large language models into GPTalk provided an opportunity to greatly enhance the application’s content with AI-generated exercises. However, this experience also revealed the limitations of using LLMs in a language learning context. On the one hand, they allowed us to generate a diverse range of content quickly, providing users with fresh and varied exercises that could adapt to their learning needs. On the other hand, we encountered challenges in ensuring the relevance and accuracy of the AI-generated content, as LLMs initially produced outputs that were not precise and deterministic enough, and later onces that were not varied enough. This required further refinement to the exercise generation prompts, and taught us the importance of carefully curating AI-generated content and implementing checks and extra parameters to ensure that it meets our requirements. It also highlighted the need for ongoing adjustments and improvements to the AI integration to maintain content quality.
* We learned the importance of breaking down a large application into many smaller components to keep the code readable and understandable for other people working on the project, and easier to debug. By dividing the project into manageable pieces, we were able to keep the codebase organized, readable, and easier to maintain. This modular approach also made it easier for us to work on each other’s code, and reduced the chances of conflicts between the branches we worked on. Additionally, it made debugging more straightforward, as issues could be isolated within specific components rather than being buried in a disorganized and bloated codebase. The use of Angular’s component-based architecture was particularly beneficial in this aspect, allowing us to create reusable and self-contained modules that could be easily updated or replaced as needed.

# Verification Testing

## Frontend Testing

**Login**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| Successful login | The user entered valid details and clicked ‘login’. | A *toastr* popup appears with the message “Logged in successfully”, the app navigates to “/pages” route. | Pass – the login was successful as expeted and the user was redirected to the home page. |
| Unsuccessful login | The user entered a wong email or password. | A *toastr* popup appears with the message “Could not login”. | Pass – the error appeared as expected and the user was not granted access to the home page. |
| Invalid email | The user entered a value not structured like an email address. | The login button is not clickable, preventing the user from attempting to log in. | Pass – the button stayed greyed out, and a red outline appeared around the email field, indicating that the email is invalid. |
| Empty email field | The user left the email input field blank. | The login button is not clickable, preventing the user from attempting to log in. | Pass – the button stayed greyed out, and a red outline appeared around the email field, indicating that the email is invalid. |
| Empty password field | The user left the password input field blank. | The login button is not clickable, preventing the user from attempting to log in. | Pass – the button stayed greyed out, and a red outline appeared around the password field, indicating that the password is invalid. |

**Register**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| Successful registration | The user filled all required input fields and clicked ‘Finish’. | A *toastr* popup appears with the message “Registration successful”, and the user is added to the database. | Pass – the message was displayed and the new user was successfully added to the database. |
| Empty name field | The user did not enter his name in the first step and clicked ‘Next’. | The user will not be able to proceed to the next step. | Pass – the button did not navigate to the next step, and a red outline appeared around the empty field. |
| Empty username field | The user did not enter his username in the second step and clicked ‘Next’. | The user will not be able to proceed to the next step. | Pass – the button did not navigate to the next step, and a red outline appeared around the empty field. |
| Invalid email | The user entered a value not structured like an email address in the second step and clicked ‘Next’. | The user will not be able to proceed to the next step. | Pass – the button did not navigate to the next step, and a red outline appeared around the email field. |
| Empty email field | The user did not enter his email in the second step and clicked ‘Next. | The user will not be able to proceed to the next step. | Pass – the button did not navigate to the next step, and a red outline appeared around the email field. |
| Invalid password | The user entered a password in the third step that did not contain every type of required character for a valid password. | The user will not be able to click the ‘Finish’ button. | Pass – the button stayed greyed out, a password hint bubble below the input field displayed the missing character types in red, and a red outline appeared around the password field. |
| Empty password field | The user did not enter a password in the third step. | The user will not be able to click the ‘Finish’ button. | Pass – the button stayed greyed out, a password hint bubble below the input field displayed the missing character types in red (all of them), and a red outline appeared around the password field. |
| Non-matching passwords | The user entered a password in ‘confirm-password’ that did not match the one in ‘password’ in the third step. | The user will not be able to click the ‘Finish’ button. | Pass – the button stayed greyed out, and a warning text appeared below the input field saying that the passwords do not match. |
| The user already exists | The user enters an email address that is already in use. | A *toastr* popup appears with the message “Account with this email already exists”. | Pass – the message was displayed and the user was not created. |

**Chat With Me**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| Language Select | The user selects a language by clicking on the flag. | The system proceeds to the chat interface, and the selected language is set to the chosen language. | Pass – The user was directed to the chat screen. |
| Chat Successful | The user sends a message. | The message is sent successfully, and OpenAI API responds with a relevant reply in the chosen language. The message appears in the conversation window. | Pass – The message was sent successfully, and the ChatGPT response appeared in the chat window. |
| Empty Message | The user clicks “Send” with the message input field left empty. | The "Send" button is disabled. The message cannot be sent if the field is empty. | Pass – The “Send” button remained disabled. |
| Multiple Messages | The user try sends multiple messages quickly in succession. | The “Send” button is disabled and not letting the user hit “Send” several times. | Pass – The “Send” button remained disabled. |
| Grade | The user clicks “Grade”. | The “Grade” button trigger a request to OpenAI API that return the grade (exp) and feedback that will be shown on a popup. | Pass – The “Grade” button triggered a request and opened a popup with the grade and the feedback. |

**Learn**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| Question Answered Correctly | The user answers a question correctly. | The system automatically moves to the next question, updates the exp, and reflects the exp gained for the correct answer. | Pass – The answer was correct, the system moved to the next question, and exp was updated as expected. |
| Question Answered Incorrectly | The user answers a question incorrectly. | The system automatically moves to the next question without updating the exp, reflecting no gain for the incorrect answer. | Pass – The system moved to the next question without updating exp, as the answer was incorrect. |
| Exp Update On Correct | The user provides correct answers and expects exp to increase accordingly. | After each correct answer, the exp increases proportionally to the user's performance, reflected in a progress update. | Pass – The exp was updated correctly after each right answer. |
| Exp No Update On Wrong | The user provides incorrect answers and expects no exp to be awarded. | After each incorrect answer, no exp is awarded, and the progress remains the same for incorrect answers. | Pass – No exp was awarded for incorrect answers. |
| Summary Display Correct | At the end of the session, the system displays the total correct answers. | A summary screen shows the total number of correct answers from the total number of questions answered. | Pass – The summary screen displayed the number of correct answers and total questions as expected. |

## Backend testing

**Lesson-generator**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| Lesson generated successfully | Lesson generated by OpenAI API successfully. | Status code 200, generated data. | Pass – Status code 200 and the generated data. |
| Error on generation | OpenAI API is down. | Status code 500. | Pass- Status code 500. |
| Lesson generated in JSON format | Lesson need to be sent as JSON object. | JSON object as response. | Pass – JSON object. |

**Chat-with-me**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| Response generated successfully | OpenAI API generates a response to the user message. | Status code 200, response of OpenAI API. | Pass – Status code 200 and the OpenAI API response. |
| Response generated in JSON format | Response need to be sent as JSON object. | JSON object as response. | Pass – JSON object. |
| Grade calculates successfully | OpenAI API grade the conversation and return the grade and feedback. | Status code 200, the grade and feedback. | Pass – Status code 200 and object of the grade and feedback. |
| Grade object generated in JSON format | Grade object need to be sent as JSON object. | JSON object as repsonse. | Pass – JSON object of the grade and feedback. |
| Error on generation | OpenAI API is down. | Status code 500. | Pass- Status code 500. |

**User-profile**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| Response sent successfully | The API return back the user data. | Status code 200, response with the user data. | Pass – Status code 200 and the user data. |
| Error on data retrieval | MongoDB is down. | Status code 500. | Pass- Status code 500. |

Verification testing played a critical role in ensuring that GPTalk met its functional and non-functional requirements before deployment. This phase of testing was designed to validate that the application’s features and functionalities worked as intended and that the user experience was smooth, reliable, and secure. Below is an outline of the verification testing process that was employed during GPTalk’s development.

## Unit Testing

Unit testing was the foundation of our verification process, focusing on the smallest parts of the application—individual functions, methods, and components. Each unit was tested in isolation to verify that it performed as expected. Using Jest for our JavaScript-based backend and Angular frontend, we wrote and executed unit tests to catch errors early in the development process. These tests were particularly valuable in ensuring the accuracy of calculations, the correctness of logic, and the proper functioning of individual components.

* **Backend**: For the backend, unit tests were written to validate the functionality of the Express.js API endpoints, the integration with MongoDB, and the security features such as Bcrypt password hashing. We also tested the interaction with the OpenAI API to ensure that the AI-generated content was being processed and stored correctly.
* **Frontend**: On the frontend, unit tests were used to verify that individual Angular components and services behaved as expected. This included testing user input validation, data binding, and use of utility functions.

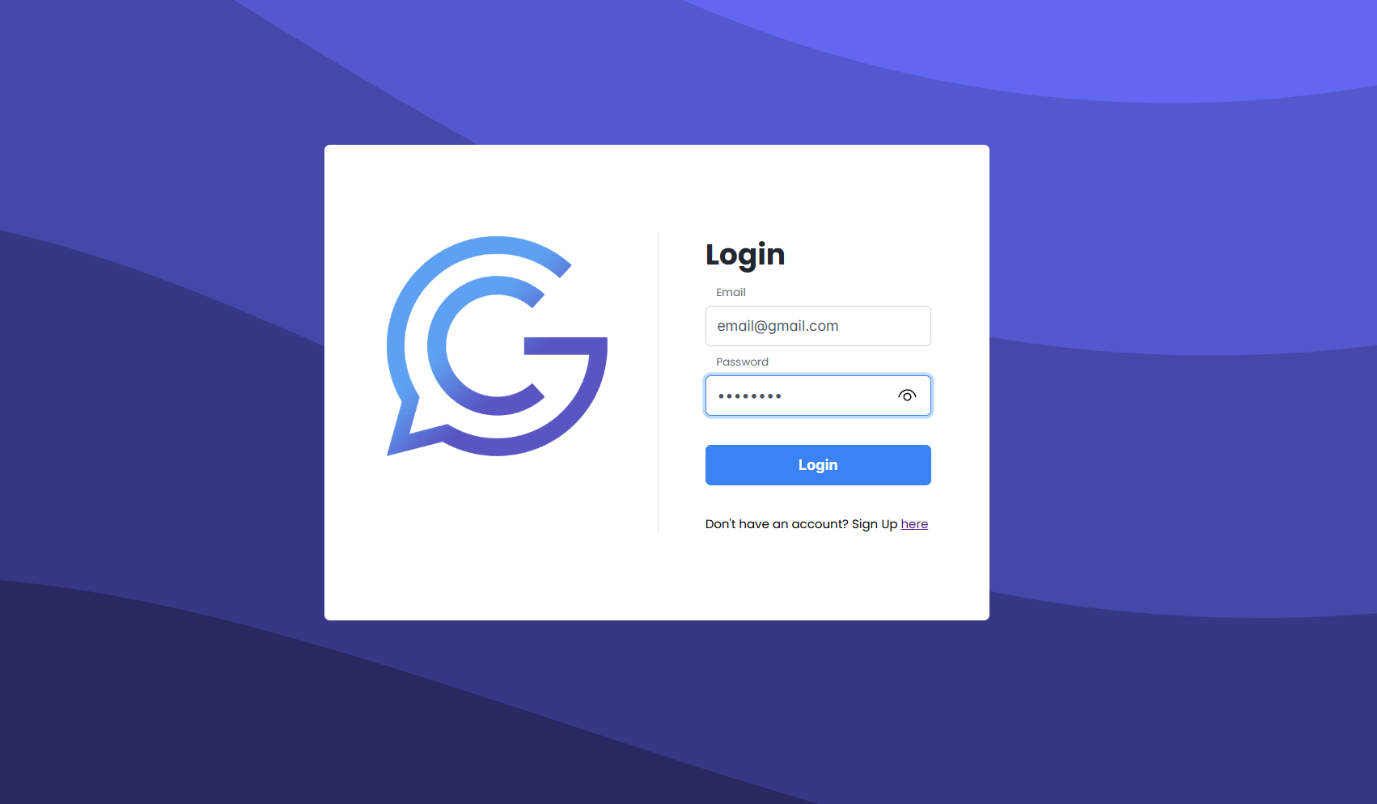
## API Testing

Given the importance of the backend API in GPTalk, thorough API testing was conducted using Postman to verify that all endpoints performed as expected. This included testing the correctness of data retrieval, submission, and processing, and the API security.

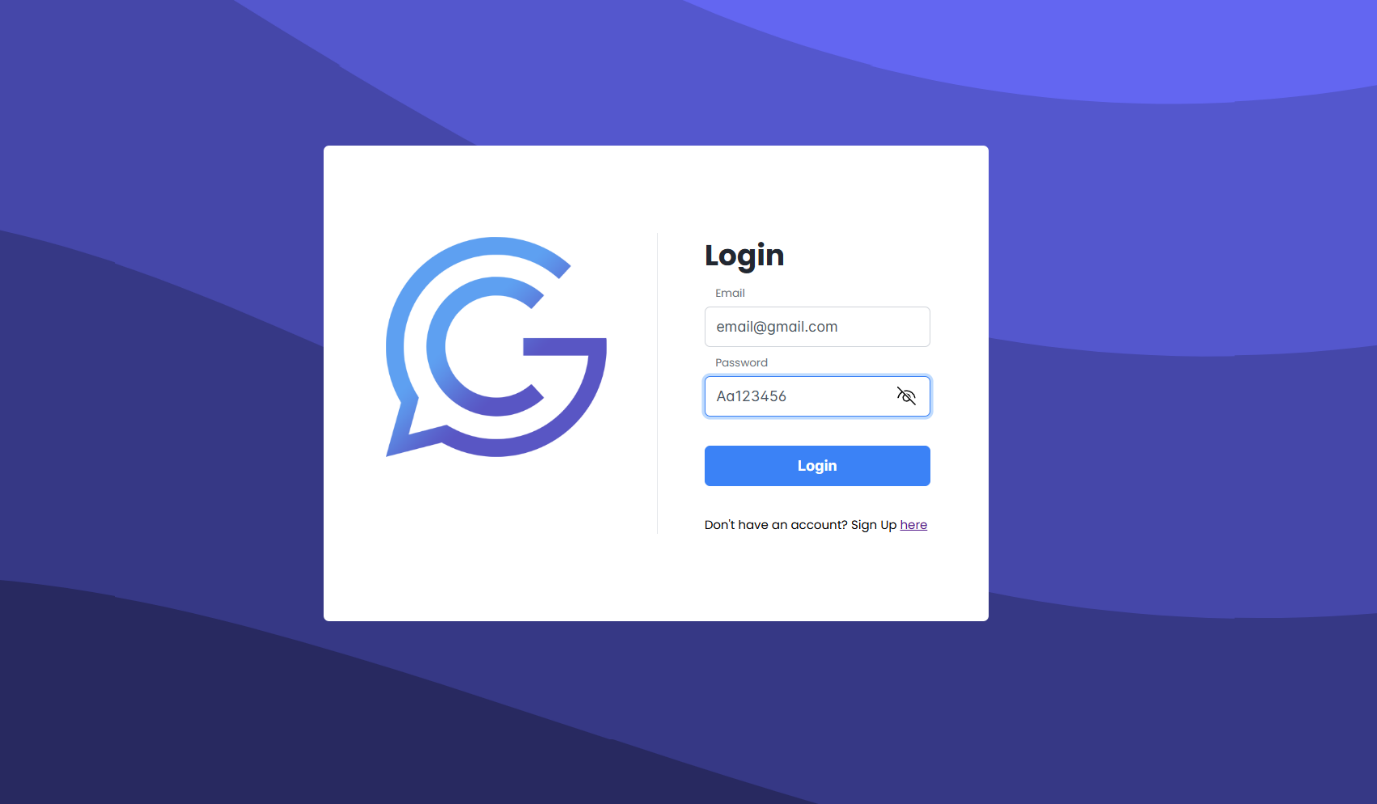
We validated that each API endpoint returned the correct data, handled edge cases appropriately, and responded with the correct status codes. This testing ensured that the backend could reliably support the frontend and that the data exchanged between the client and server was accurate.

# User Guide

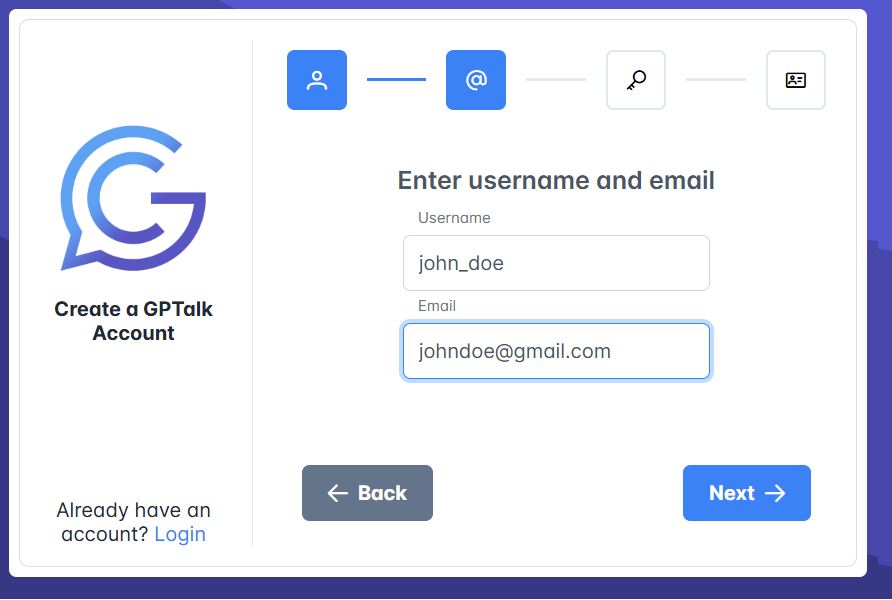
## Login and Registration

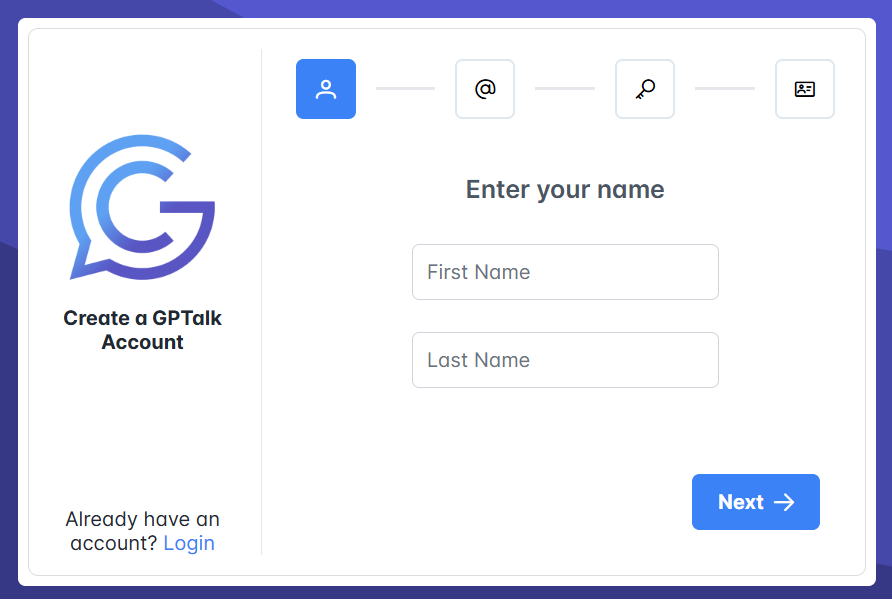


Opening the website will land you on the login screen. Enter your email and password to access sign in, or click on the sign-up link to register a new account.

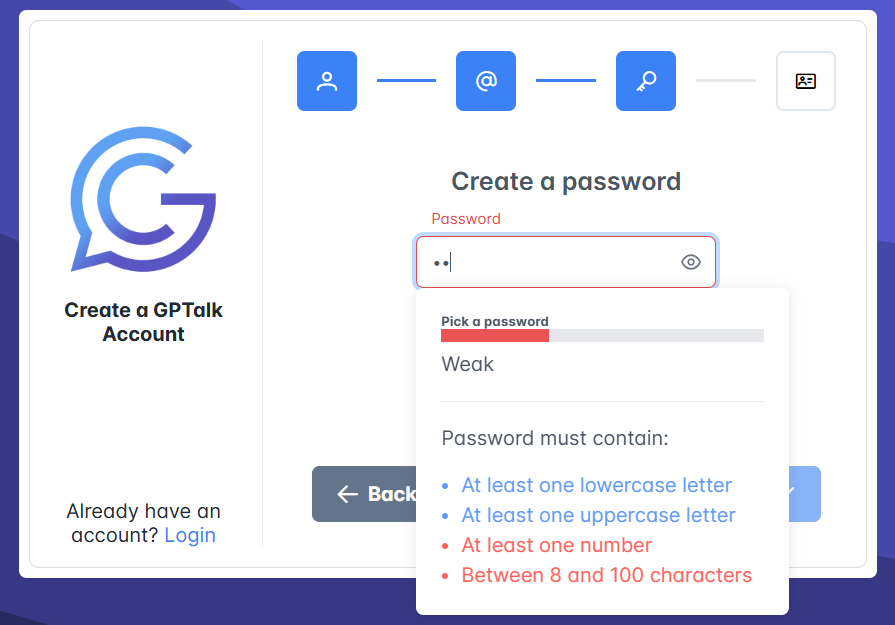


When entering a password, you can click on the eye icon to view the password you are entering.



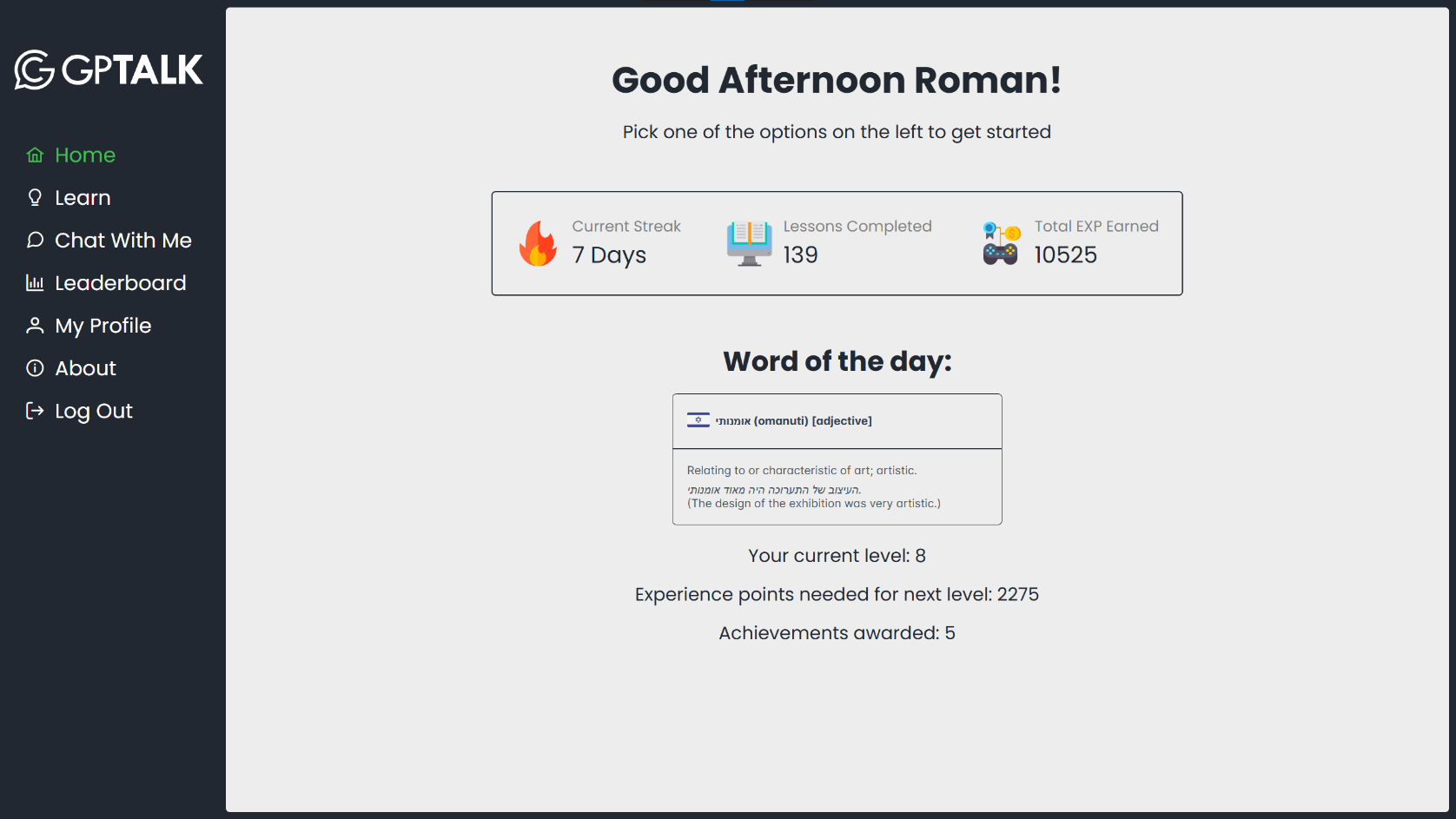


Enter the required details in each step, click next when you filled all fields in the current step.



In the final step entered a password and its confirmation. When entering a password, relevant information will appear below the input form showing you your password strength, and all character types that it must contain to be considered valid.

## Home Screen



**1**

**2**

**3**

**4**

This is the initial screen you will see when you log into the website. On it you will see the following:

1. **General statistics** of the currently logged in user. A more detailed view is available in the My Profile screen.

2. **Word of the day** – a random word in one of the available languages, including transliteration, word type, definition, example usage, and translation. A new one will be generated on a daily basis.

3. **Extra stats** indicating your current progress

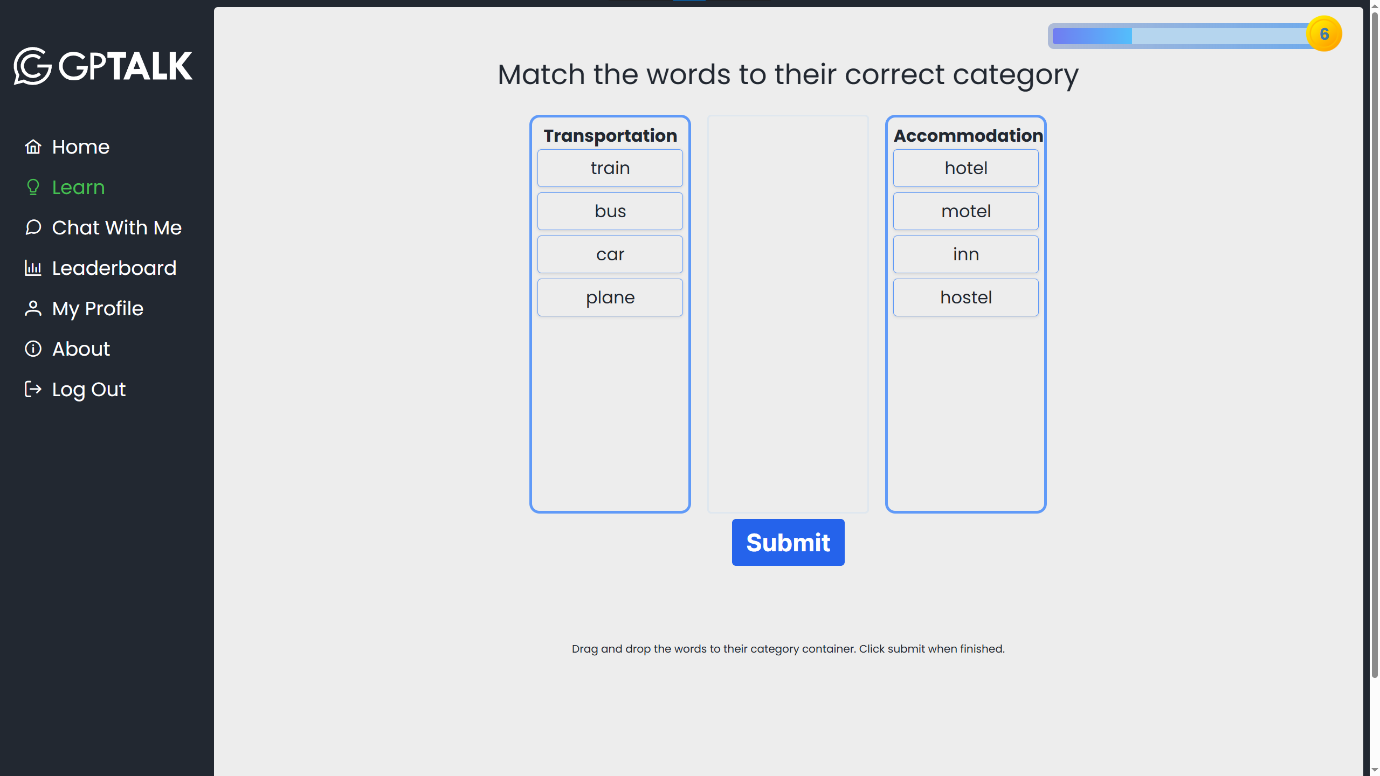
4. **Sidebar** – a menu with all the options available in the website.

## Learn

One of two main methods of interacting with the application. Allows the user to solve various exercises generated based on the chosen language and the user’s rank in that language.

When starting the Learn mode, you will have to choose the language of the generated exercises. Your progress in each language is tracked separately and can be viewed in the My Profile screen.  
If the chosen language is English, the answers in the exercises will usually have to be entered or selected in Hebrew. Otherwise, the answers will have to be entered or selected in English.

Whenever you are in Learn mode, you will see a few see several pieces of information:



**1**

**2**

**3**

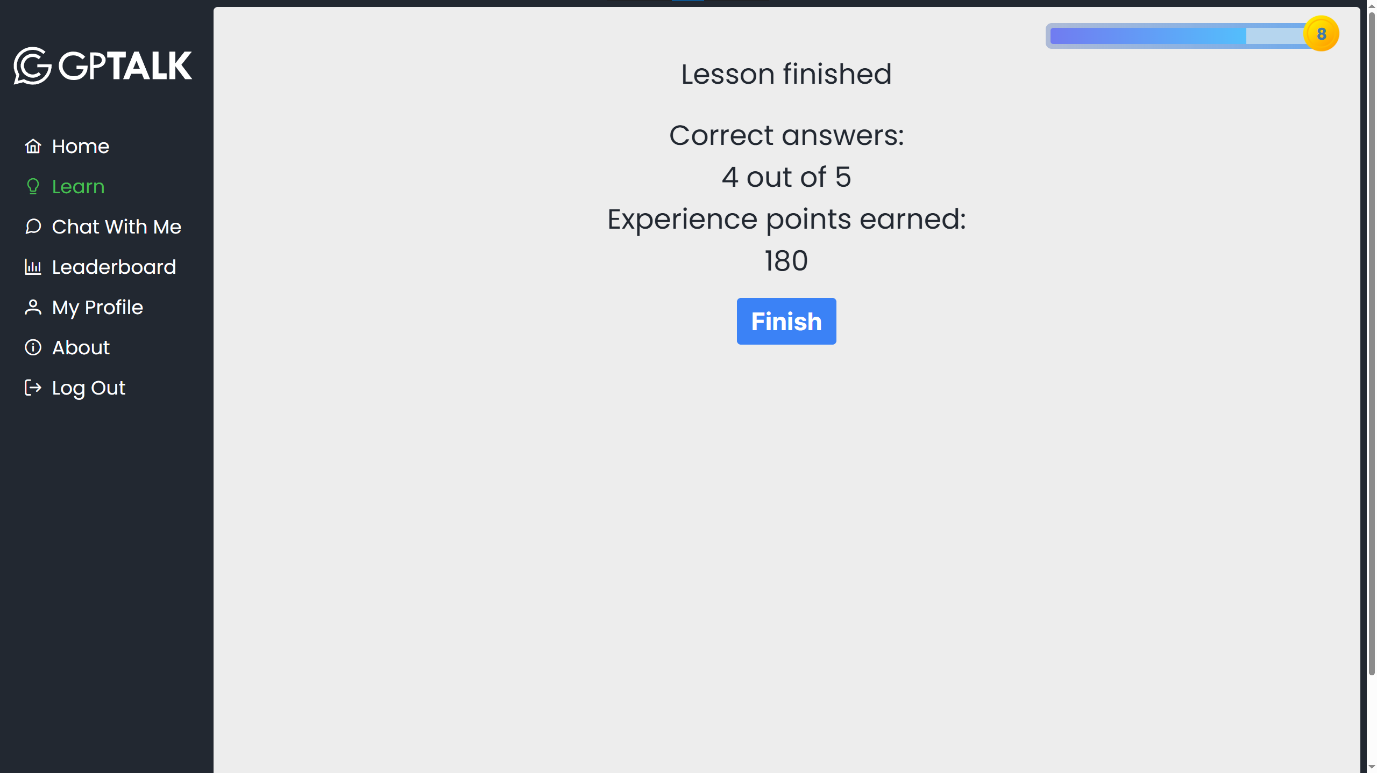
1. **Exp bar** – tracks the user’s progress towards the next level. The number on the right shows the user’s current level.



Hovering the mouse over the exp bar displays the numerical amount of experience points, and the amount left to get to the next level.

2. **Exercise heading** – gives the general goal of the current exercise.

3. **Exercise-specific instructions** – shows more detailed instructions explaining the goal of the current exercise.

Learn mode consists of lessons, which are groups of multiple randomly generated exercises.

When finishing a lesson, a results screen will appear, containing the amount of correct answers in the lesson and the exp reward.

The are several exercise types available in GPTalk’s Learn mode:

### Fill in the Blank

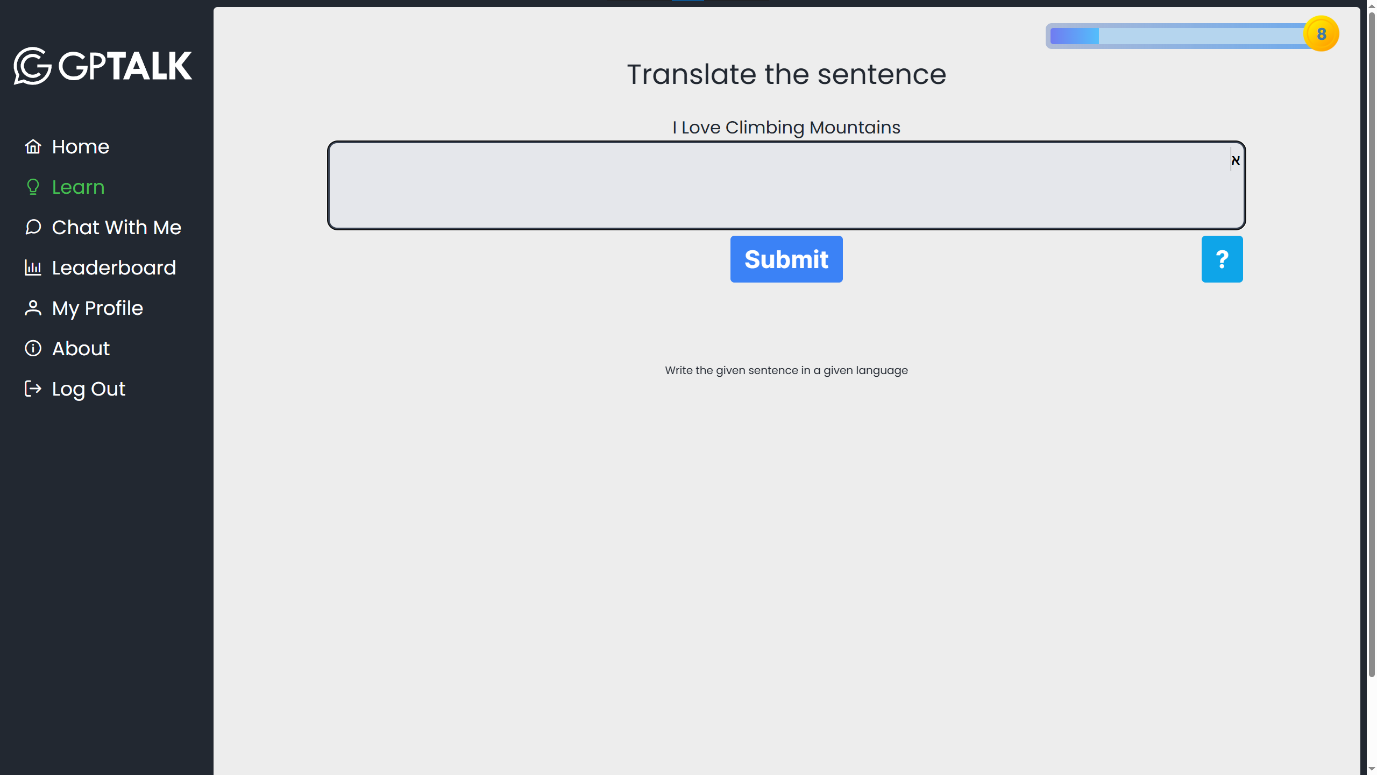


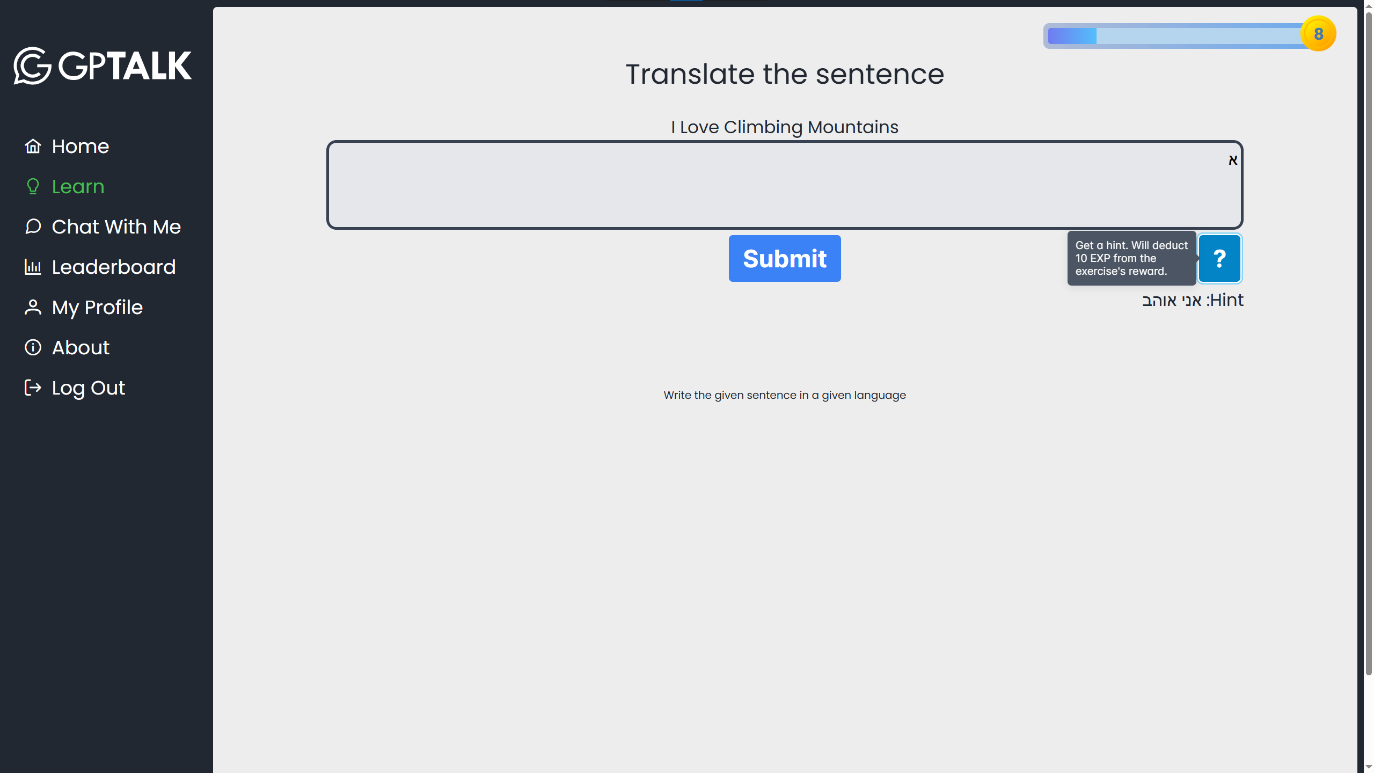
A partially completed sentence is displayed on the screen. The user has to choose the correct word that fits in with the rest of the sentence at the indicated position.

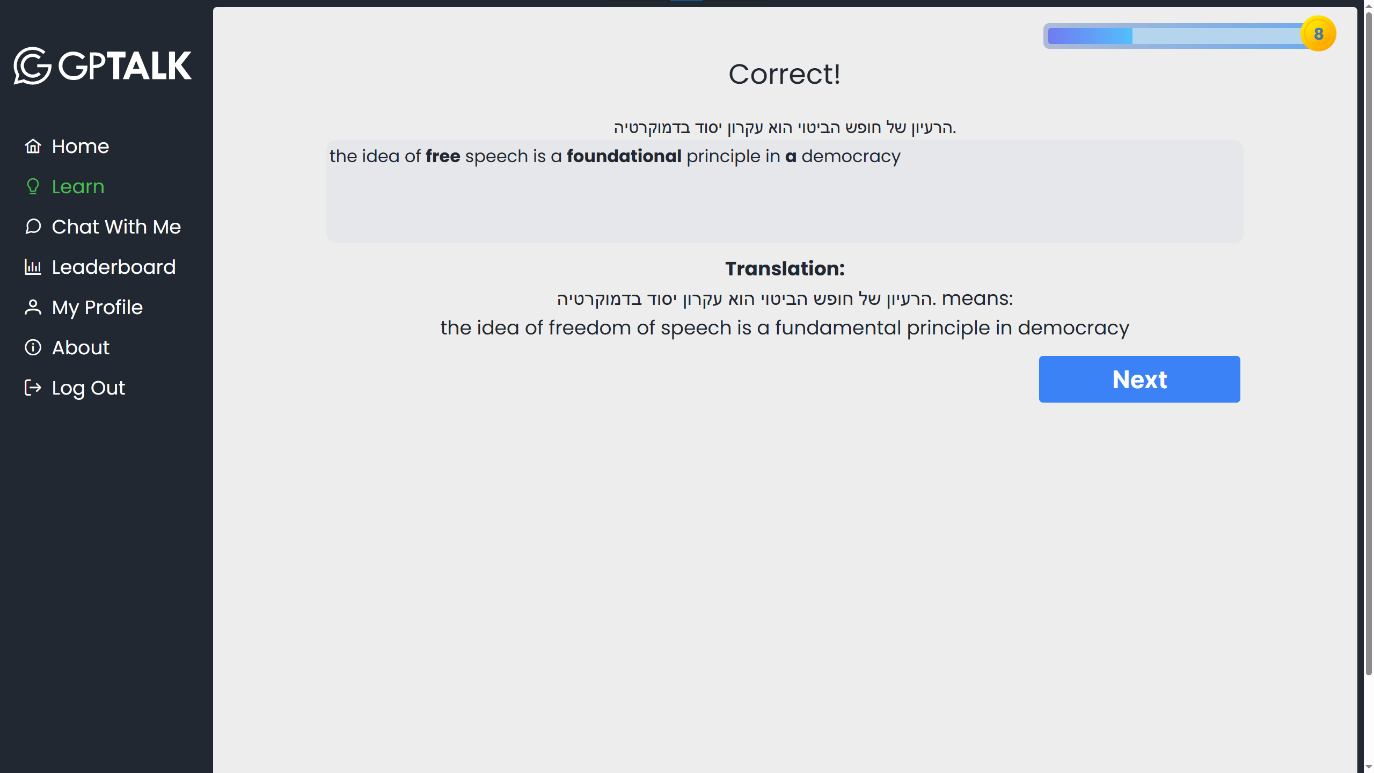
### Translate the Word

A word in the language the user chose is displayed and the user has to choose the correct English translation out of the available ones.

### Translate the Sentence



A sentence is given, and the user has to translate it to the language he chose by typing it in the input area. Since this exercise can be quite challenging, the user can get a word from the answer by clicking the hint (marked with ‘?’) button.

Every hint used reveals another word from the answer, but each one also incurs a penalty of 10 exp from the exercise’s reward.

When the user submits his answer, the application calculates the distance between his answer and the actual translation. If the answer is close enough, it will still be considered correct even if it is not exact, however a penalty will be applied to the amount of awarded experience points based on the number of mistakes in the user’s answer.   
In addition to the actual translation, the wrong words will be highlighted in the user’s answer.

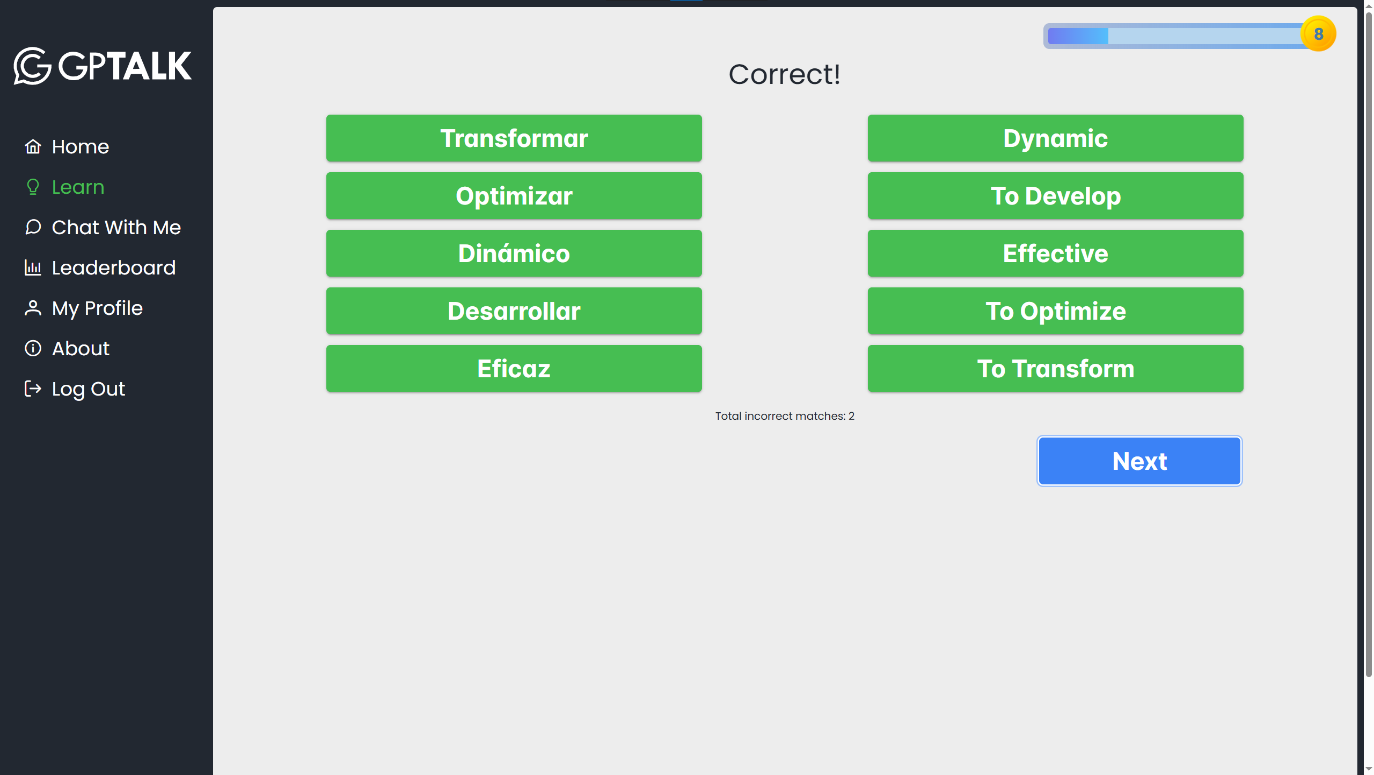
### Complete The Conversation

A chat between two participants is displayed on the screen. The conversation prompt is given in the top speech bubble, and the user has to choose the correct reply out of the available ones to put into the second speech bubble.  
The user can also practice his typing skills in the language he is learning by directly typing the reply in the second speech bubble, which doubles as an input field.

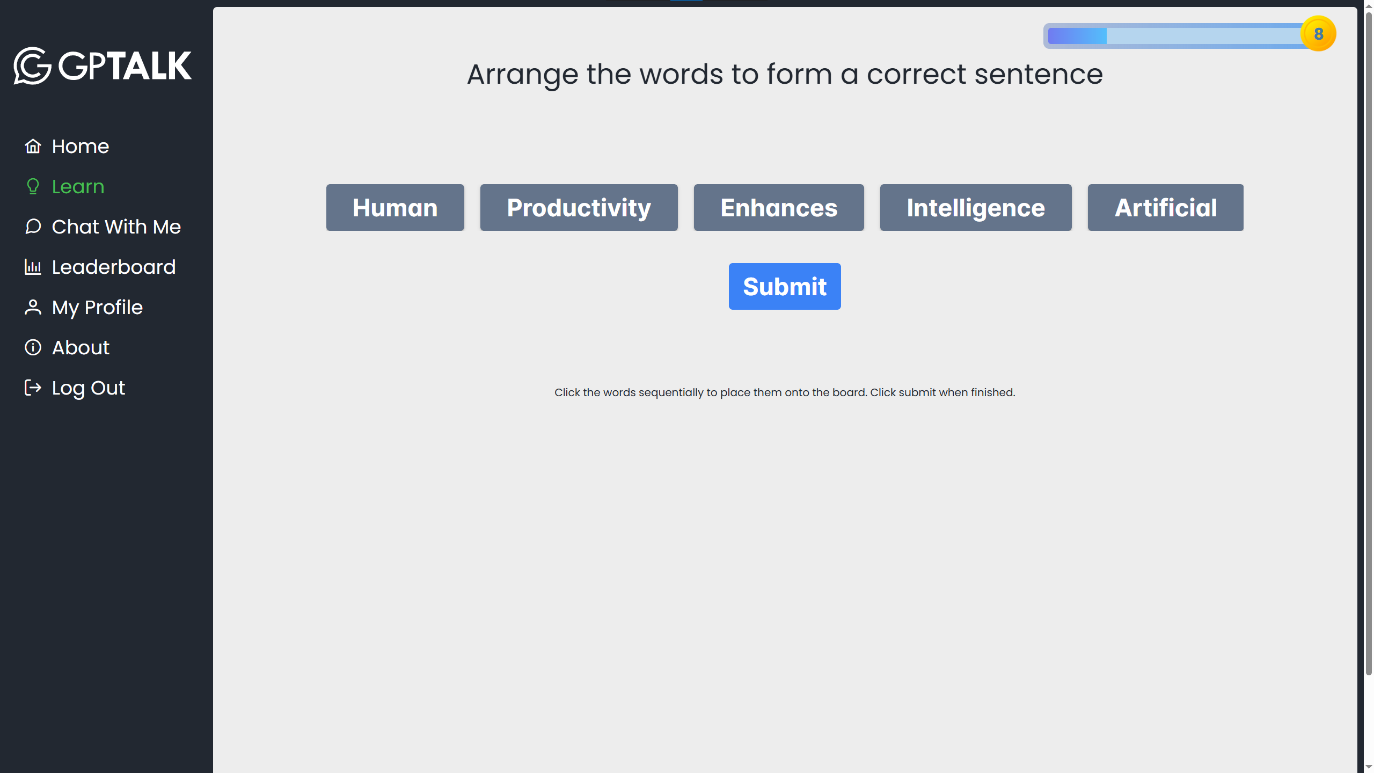
### Match The Words

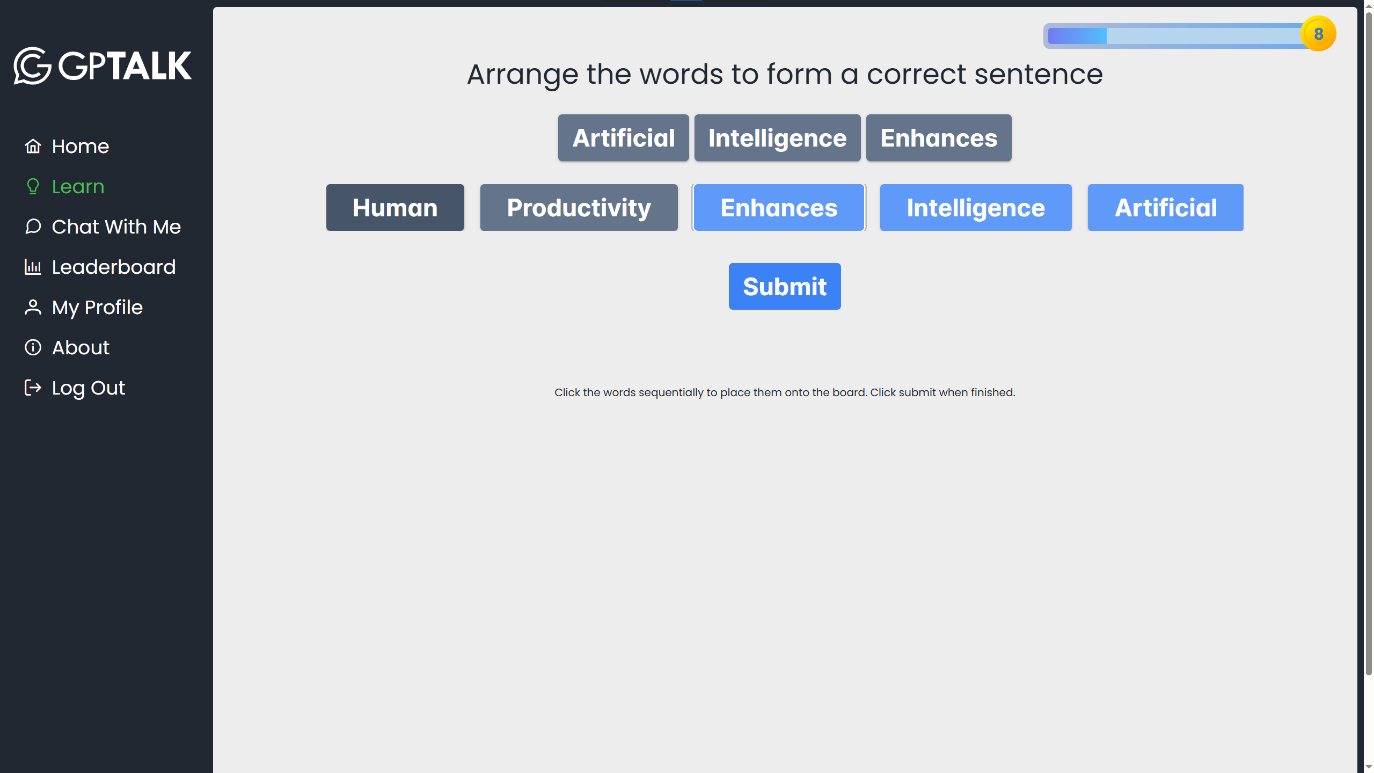
The user is given two randomly ordered word columns and he has to match each word to its equivalent in the second language by clicking on a word in one column and then on a word in the other one.

Matching incorrect pairs will give a penalty of 10 exp from the final reward for each incorrect match. Once the exercise is over, the total number of wrong matches will be displayed.

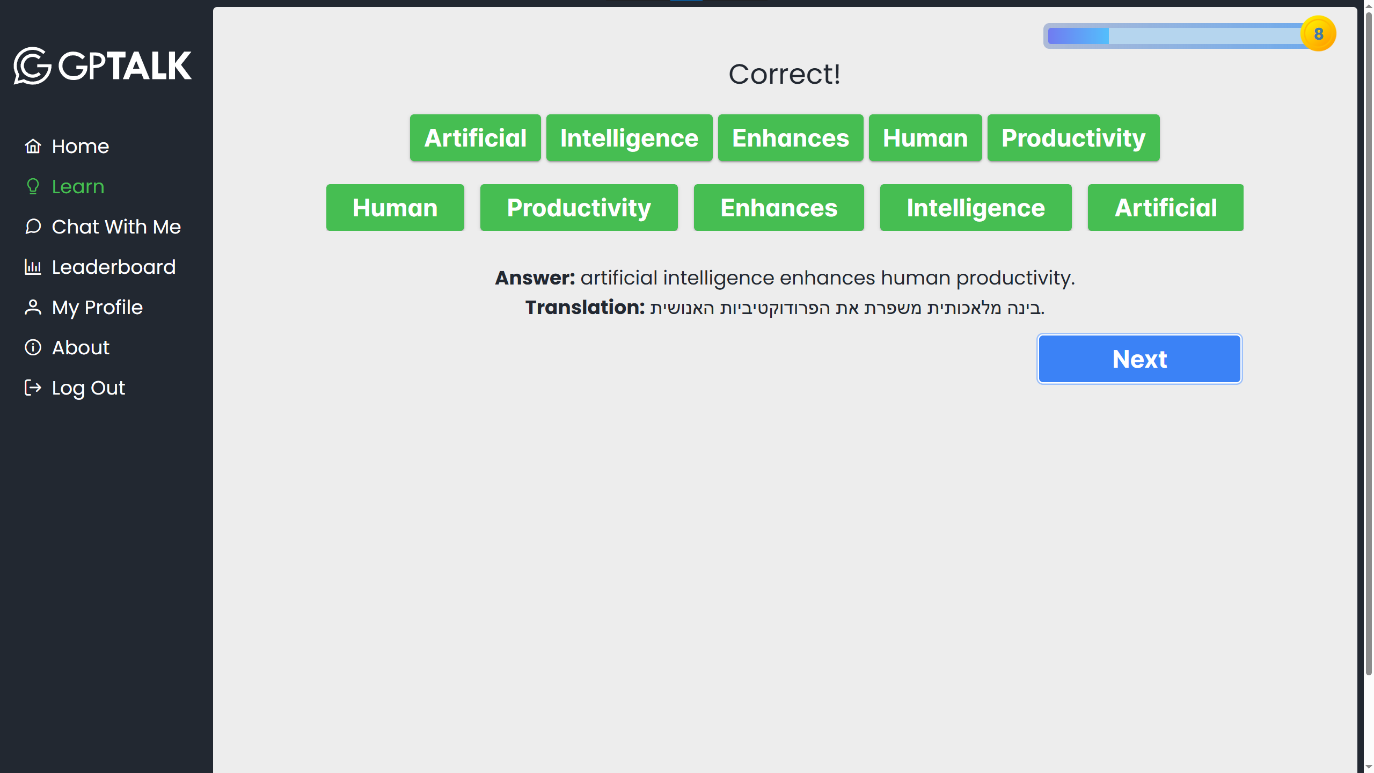


### Reorder Sentence

Several buttons containing words will appear on the screen. The user has to arrange them in the correct order to form a proper sentence.



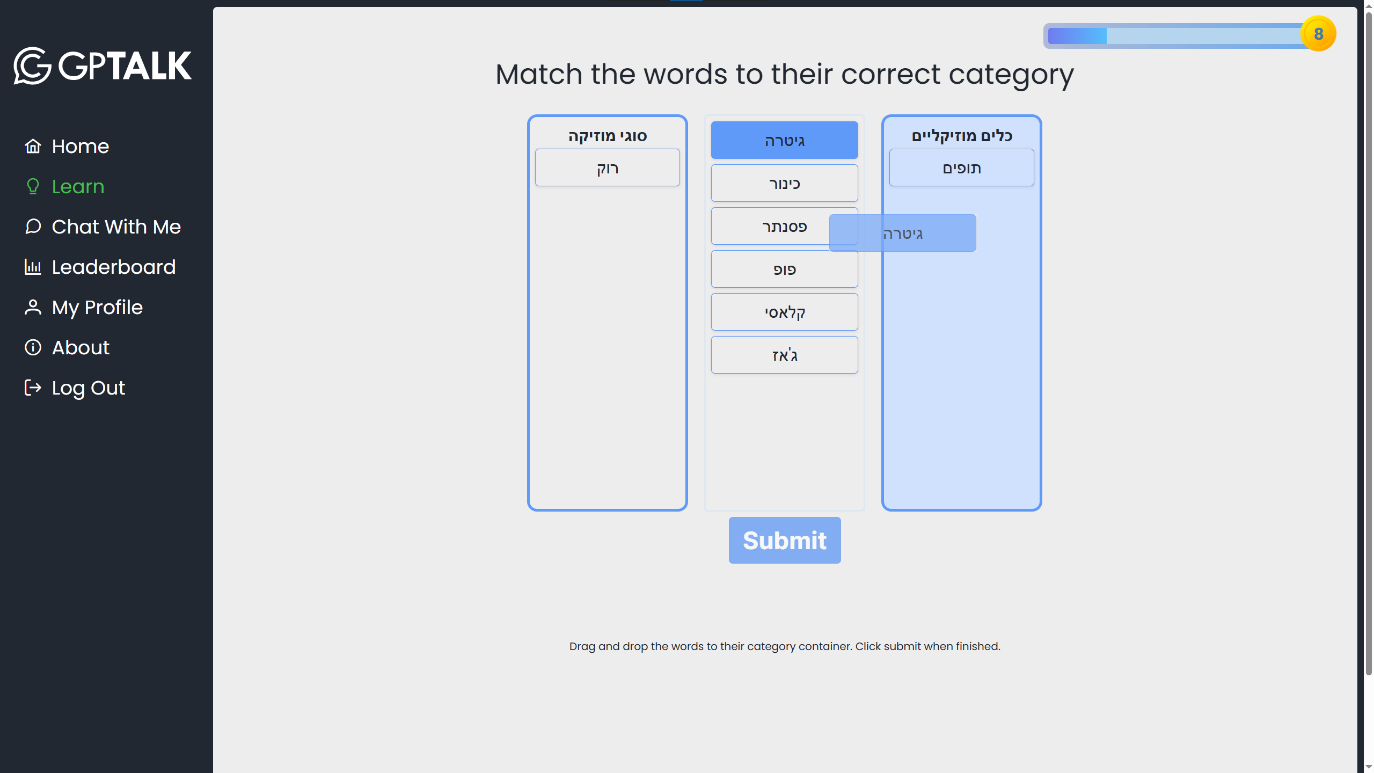
Add a word to the formed sentence by clicking on the words in the list. Once added, the word appears above the initial list. You can remove a word from the formed sentence by clicking on it again in either row.  
When finished, click Submit.

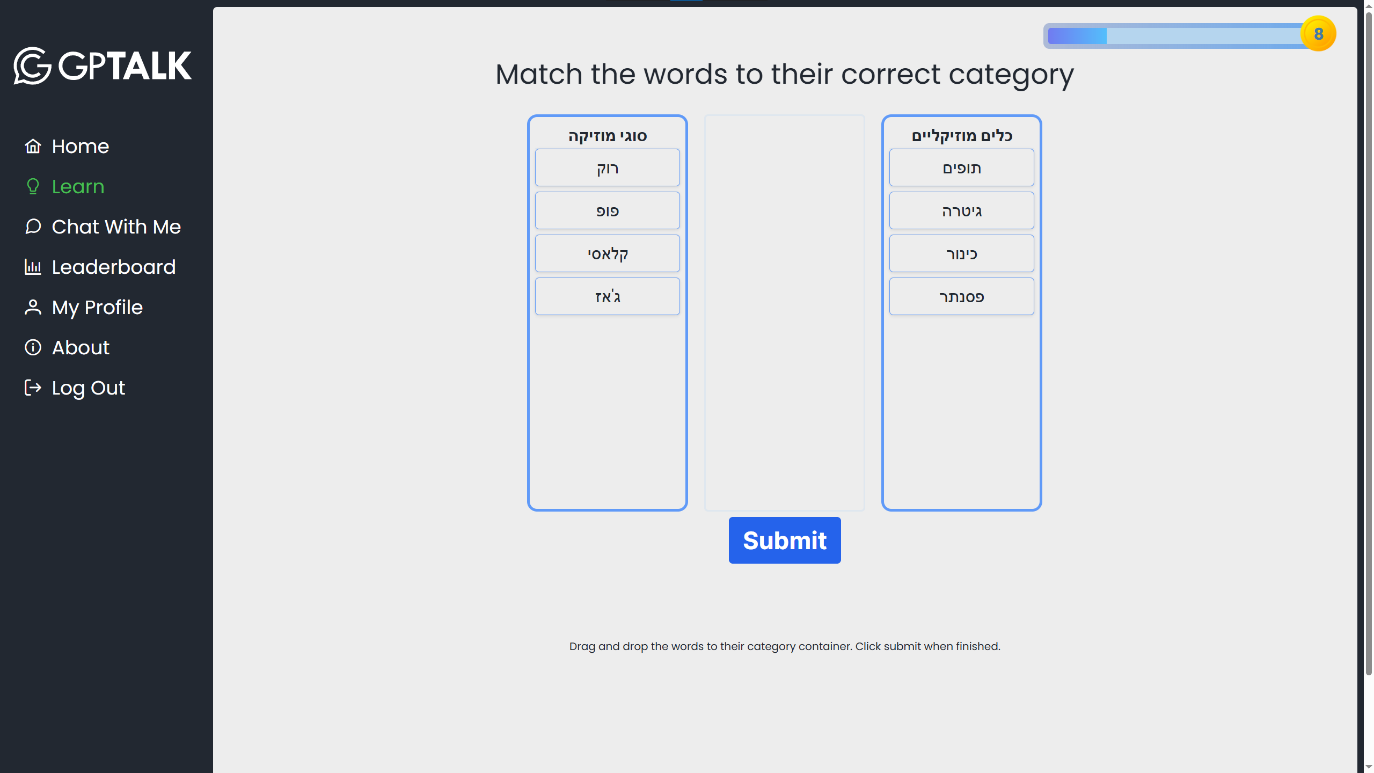


Whether the formed sentence is in the correct order or not, the actual answer and the translated sentence will appear once the user submits his answer.

### Match The Category

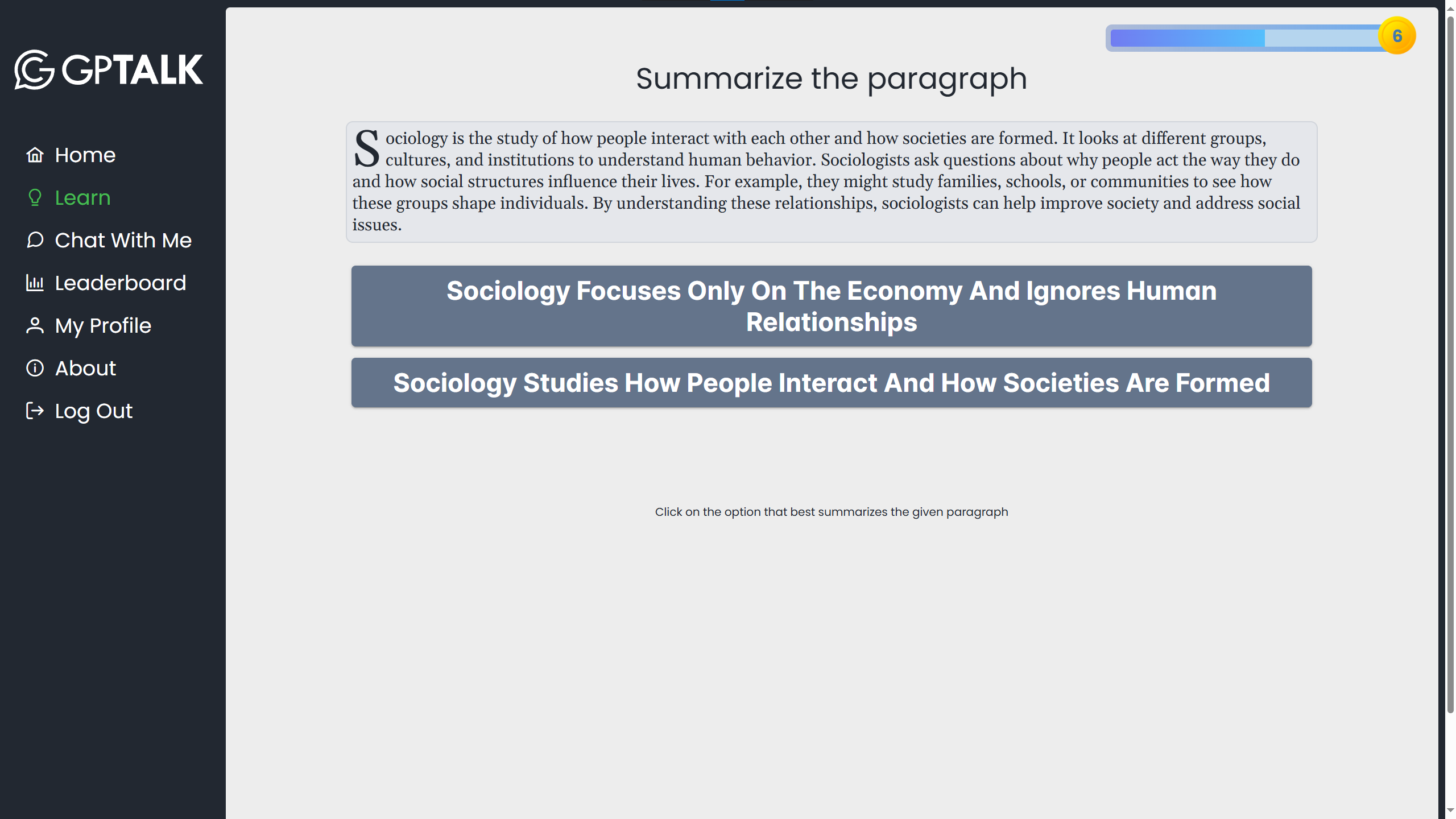
A list of words is available in the middle of the screen. The User has to match each word to the correct category of the available two.

Move a word to a category by dragging and dropping it into one of the containers.



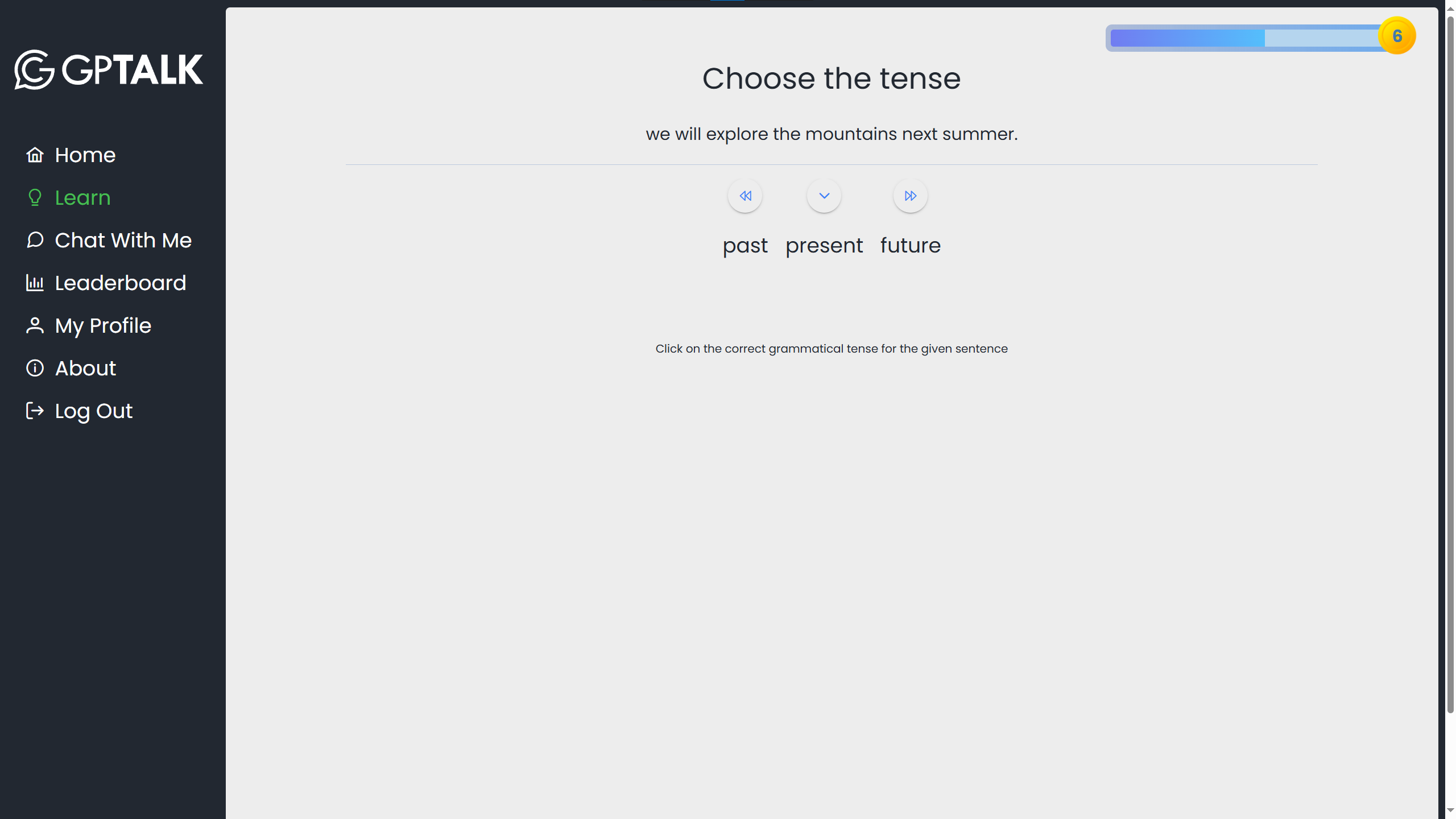
Once each word from the middle list has been placed in a category, the answer can be submitted.

### Summarize the Paragraph



A paragraph on a random topic is given. The user has to choose the option that best summarizes the paragraph.

### Choose the Tense

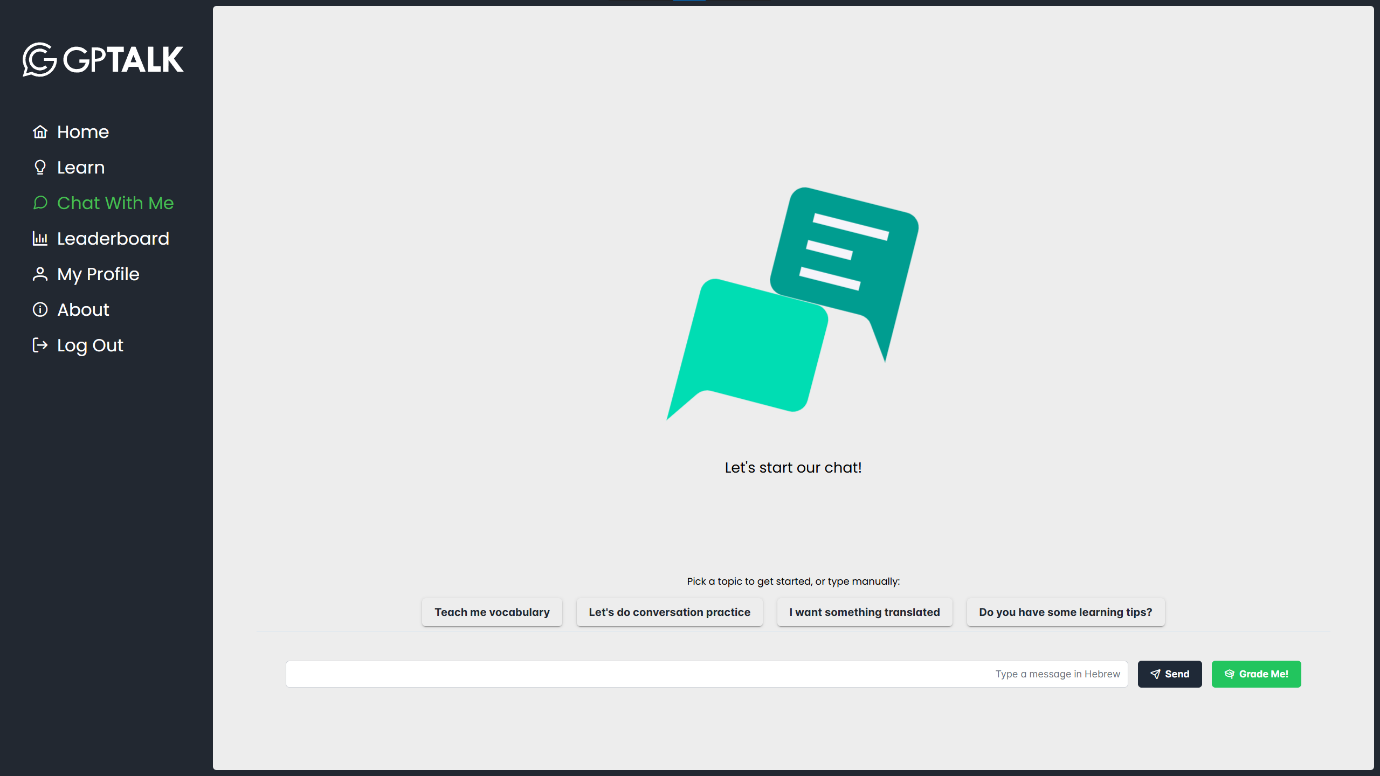


A sentence is given in the chosen language. The user has to determine whether its grammatical tense is *past*, *future*, or *present*.

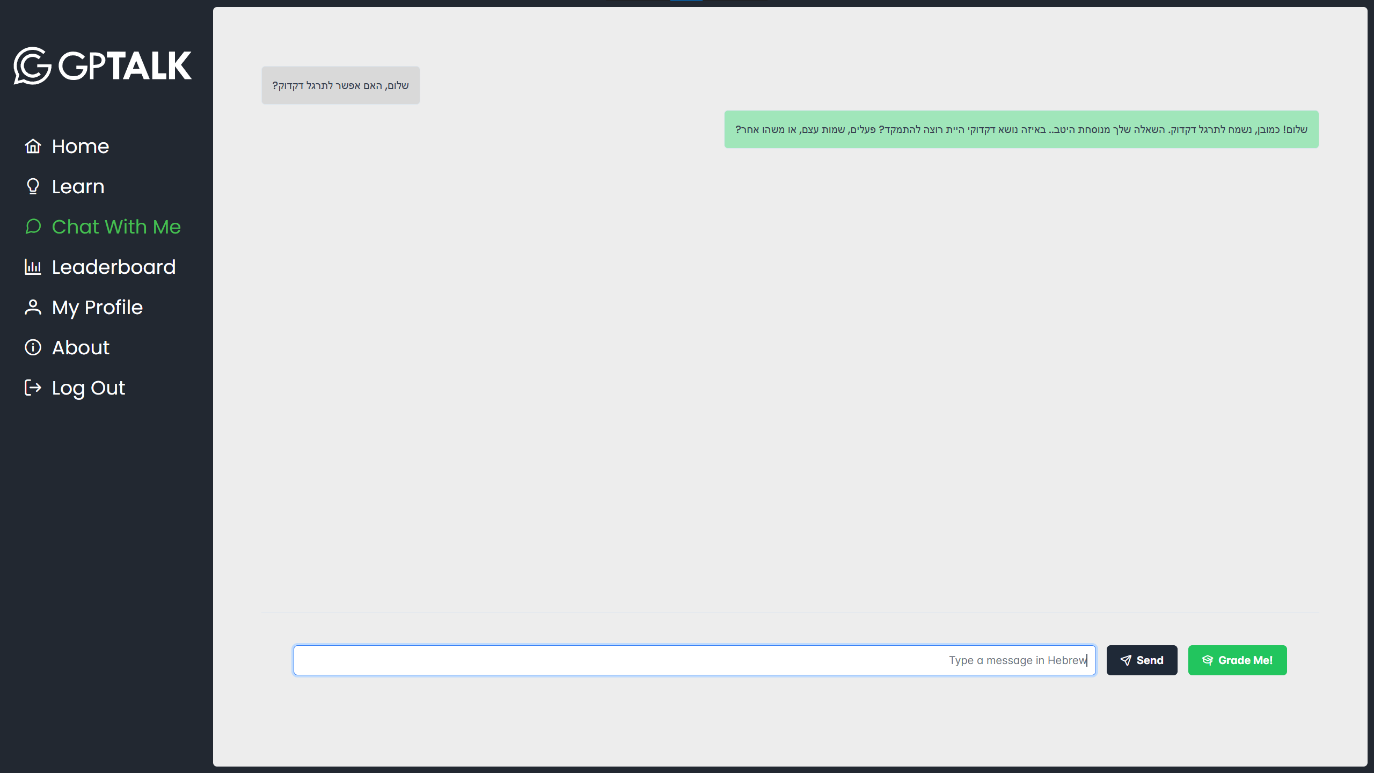
## Chat With Me

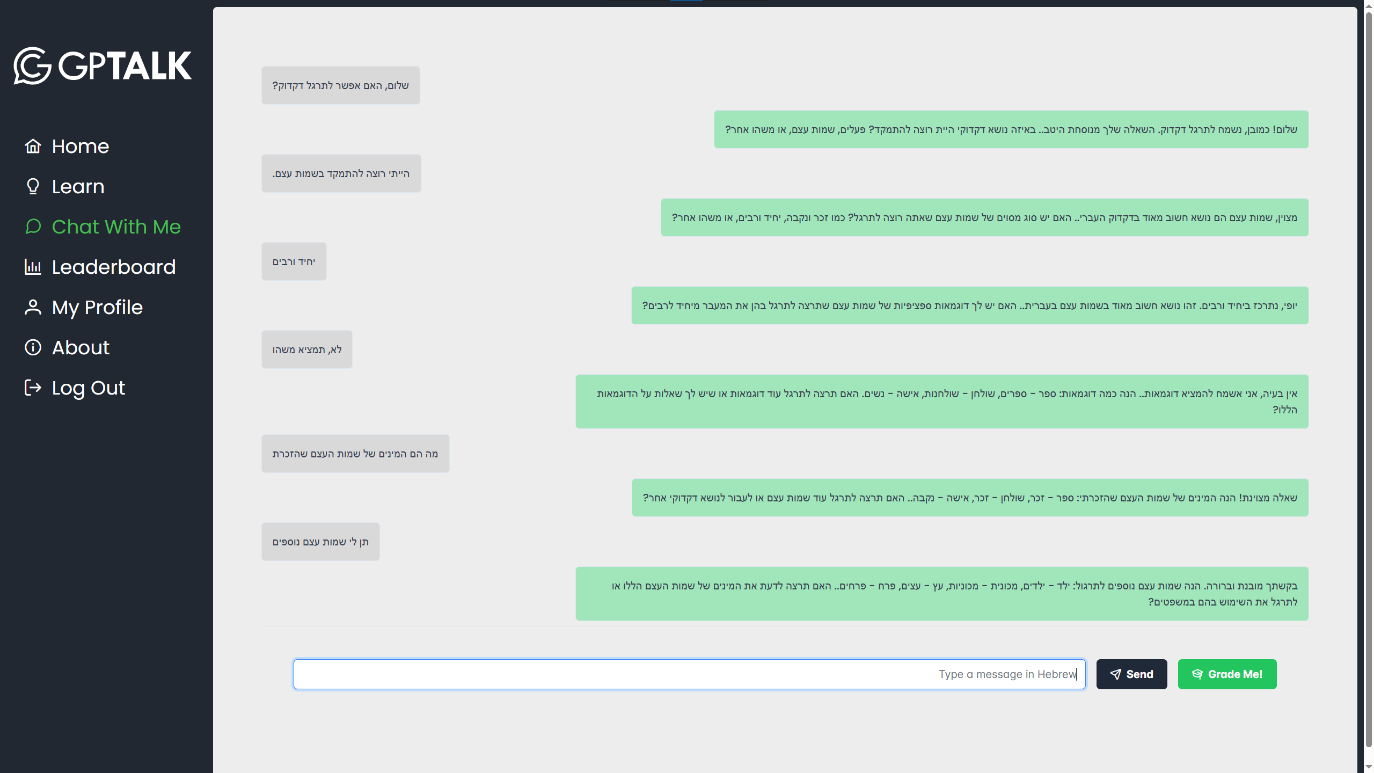
The second main method of interacting with the application is Chat with Me. This mode can be used to simulate a conversation with a person, practice recently learned concepts, get language learning tips, learn new words, grammar, and whatever else the interests the user in the world of language learning.

Like in learn mode, you will have to choose a language you want to focus on before you begin.

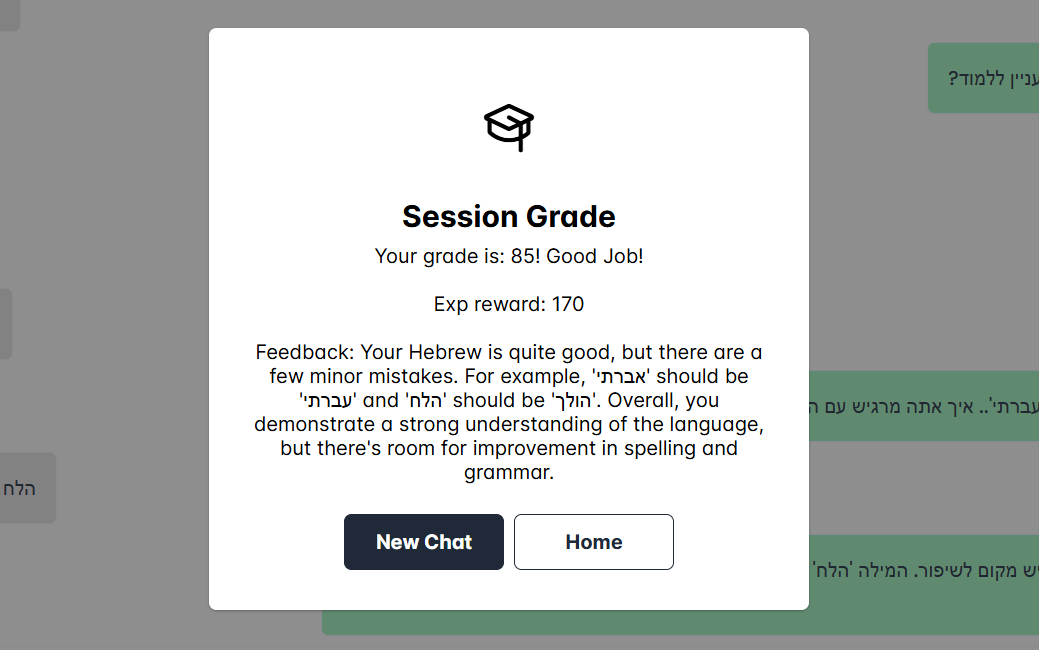


Enter a starting message in the language you chose (or in English if you are not familiar with it) and hit the Enter key or press Send to send the message. You can also start the conversation by picking one of the suggested topics above the input area.

The chatbot will reply based on the user’s prompt, and ask a follow up question relevant to the conversation subject, aimed to clarify the user’s request or to give them ideas for more questions that they can ask.

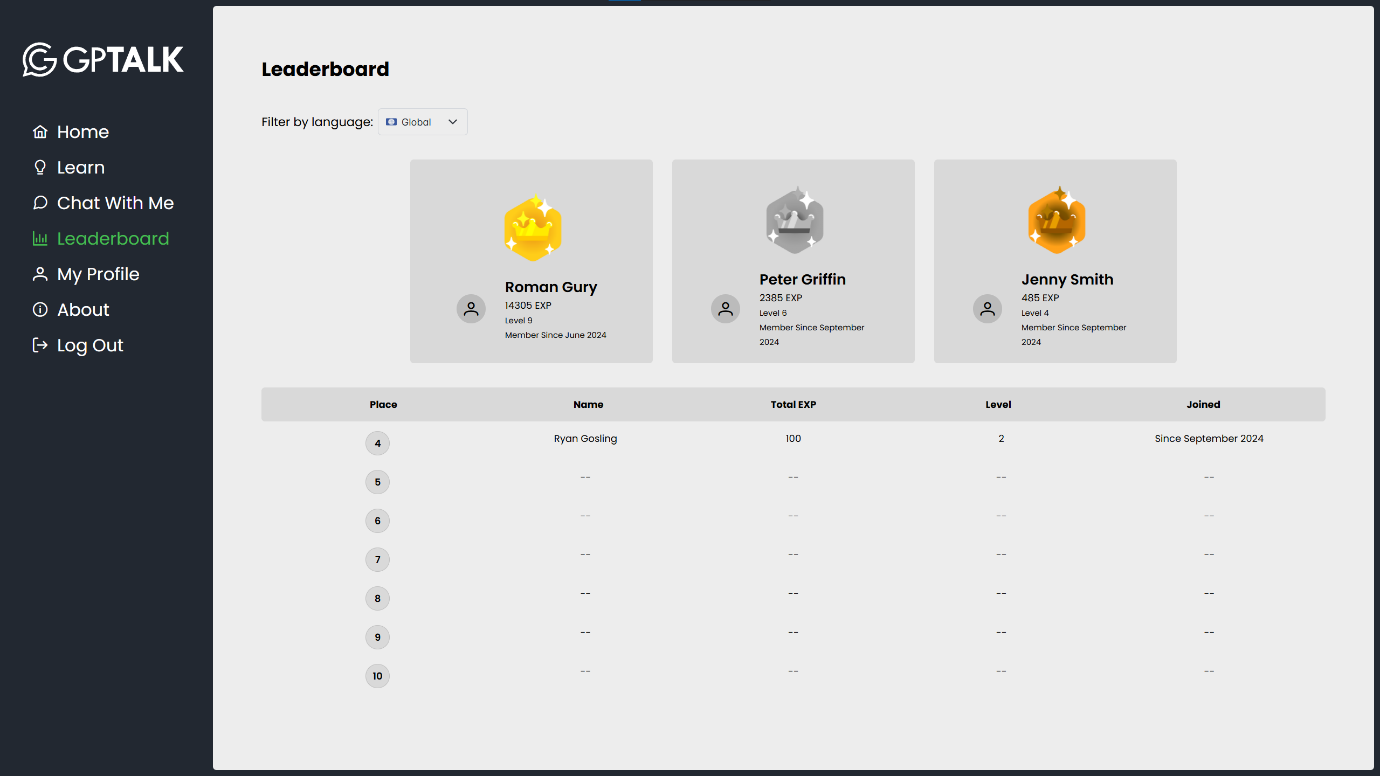
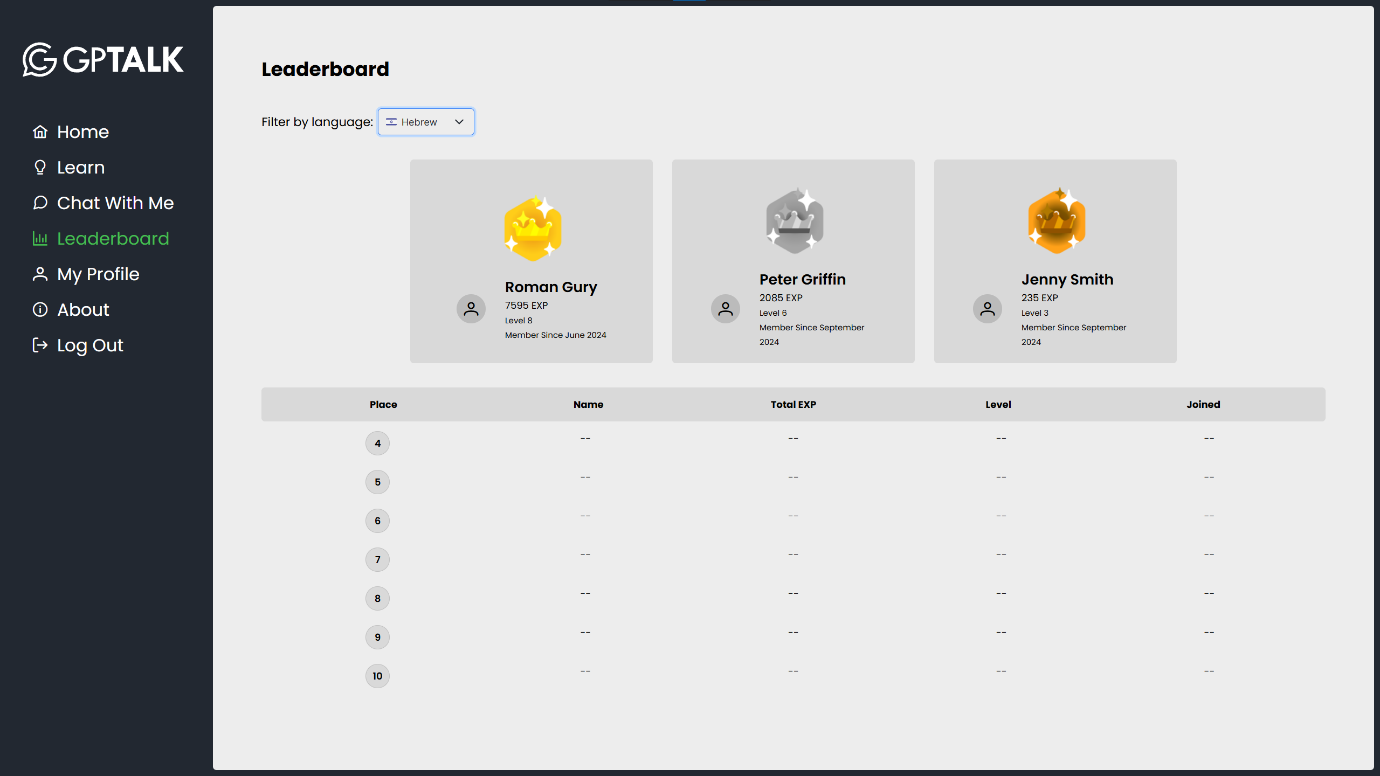


You can choose to end the chat session whenever you like by clicking the Grade Me button.



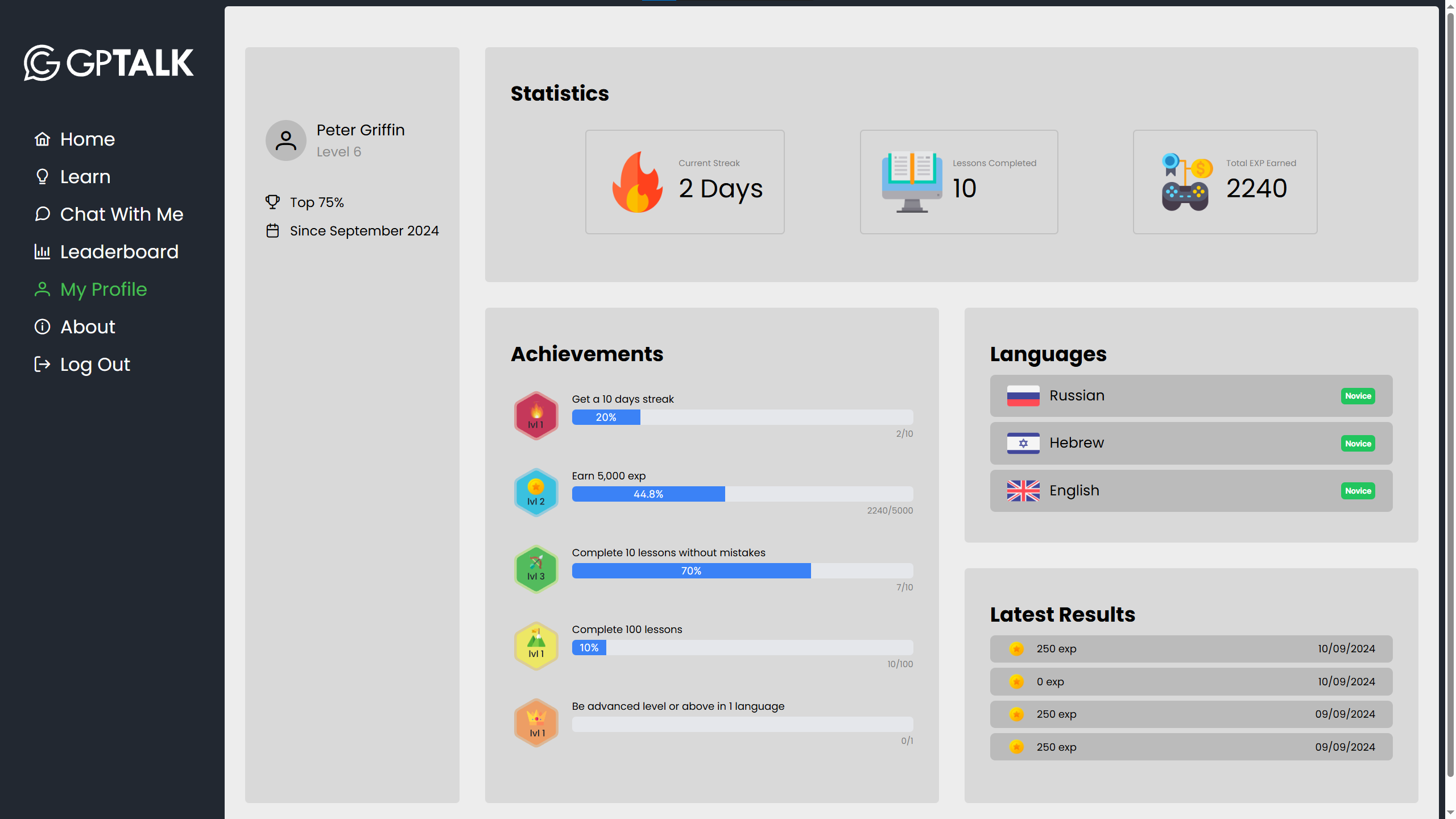
Once the session is finished, the chatbot analyzes the conversation in terms of language-related concepts like grammar and spelling, and gives the user a grade, and some feedback based on their performance. The higher the grade, the larger the exp reward for the session. Longer conversations give a larger amount of experience points.

## Leaderboard

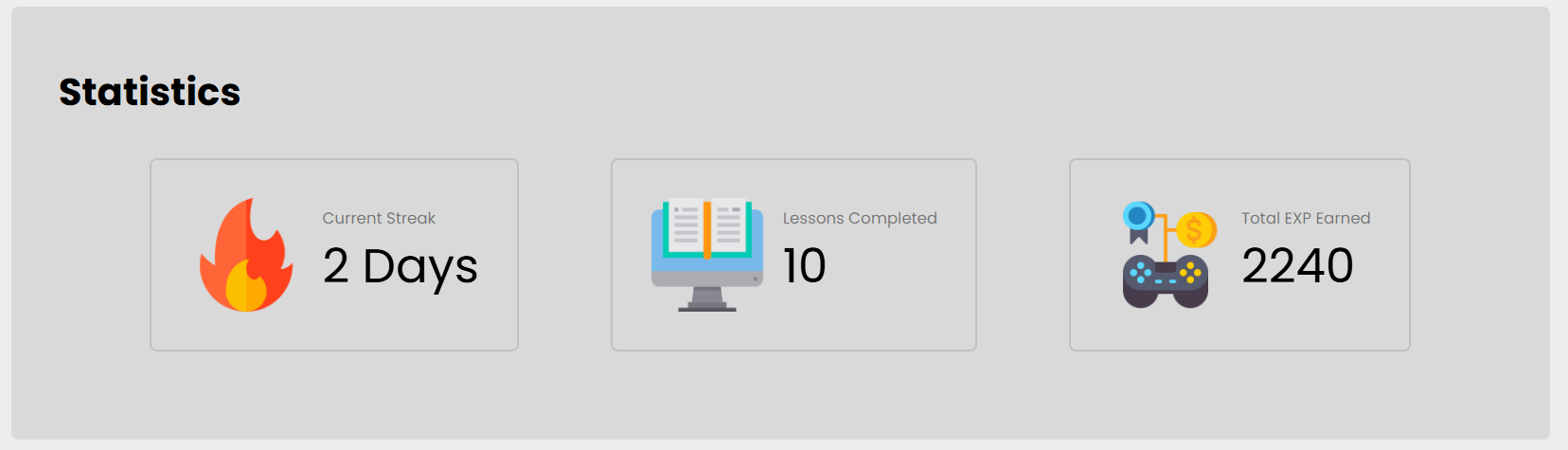
The user can see the names and stats of the top 10 GPTalk users. By default, the users are sorted by their total accumulated experience points. To filter the leaderboard by a specific language, select a language from the dropdown menu.

The leaderboard will then be updated with the top exp numbers in the chosen language.

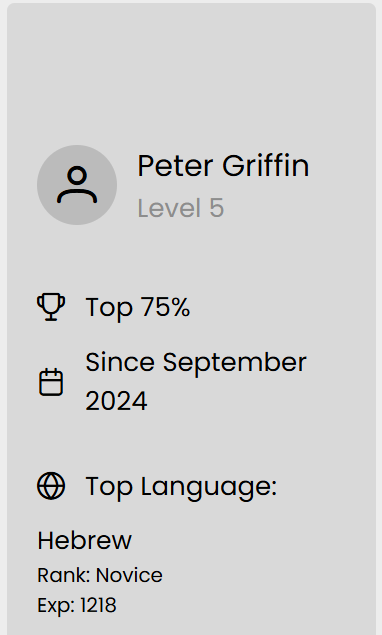
## My Profile

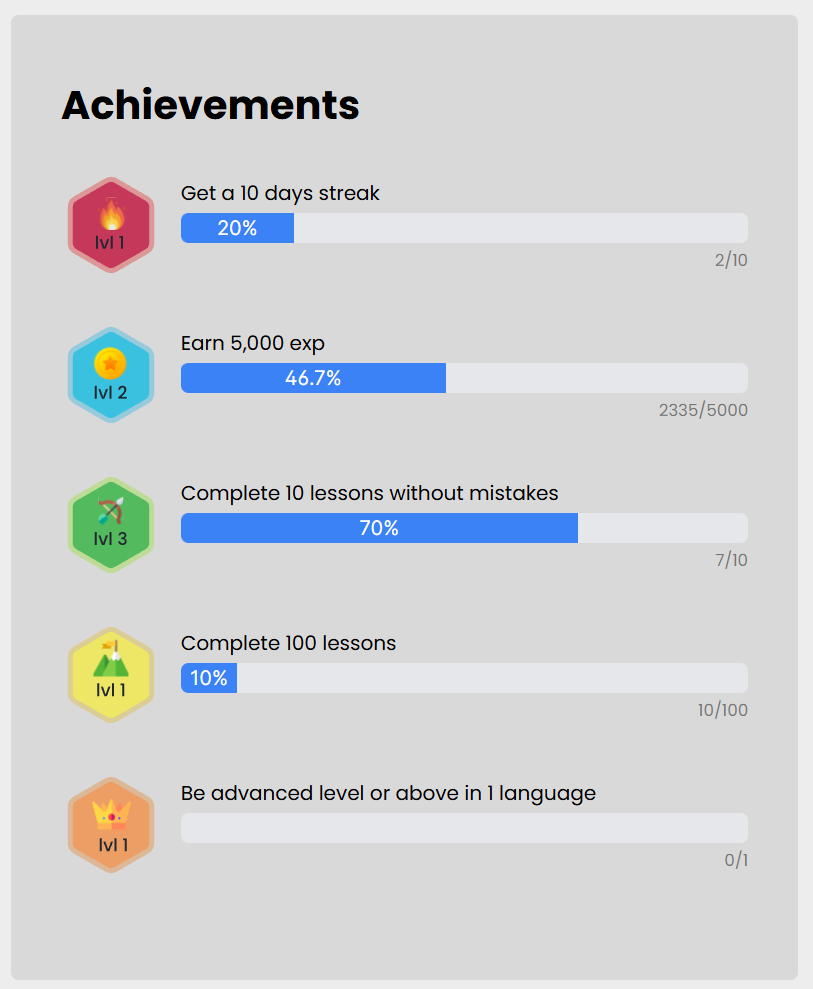


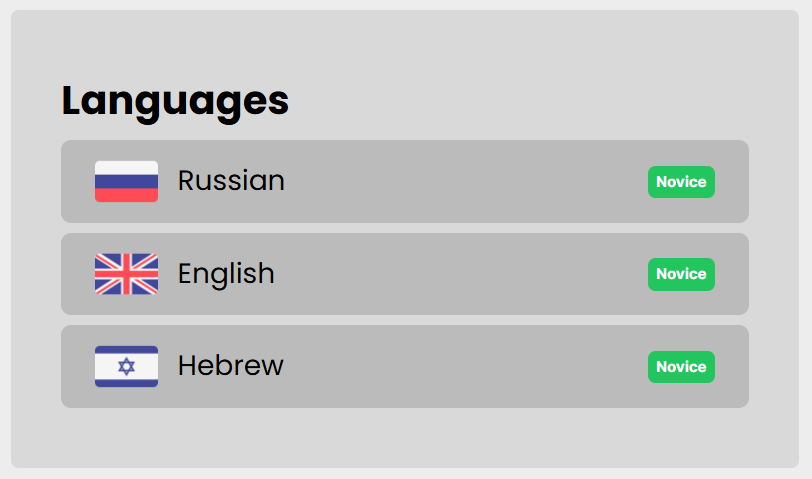
The My Profile screen shows a detailed view of the user’s personal statistics and a variety of information related to his use of the website.

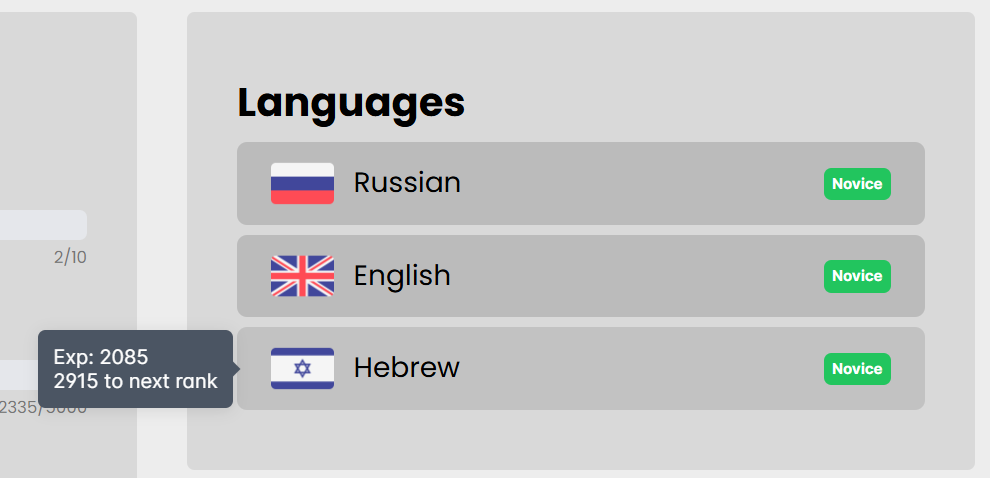


**Statistics** – general user stats: the *current streak* is the number of consecutive days the user has interacted with the website, *lessons completed* counts the user’s completed lessons in Learn mode, and *total exp earned* is the total amount of experience points the user has acquired in all modes.

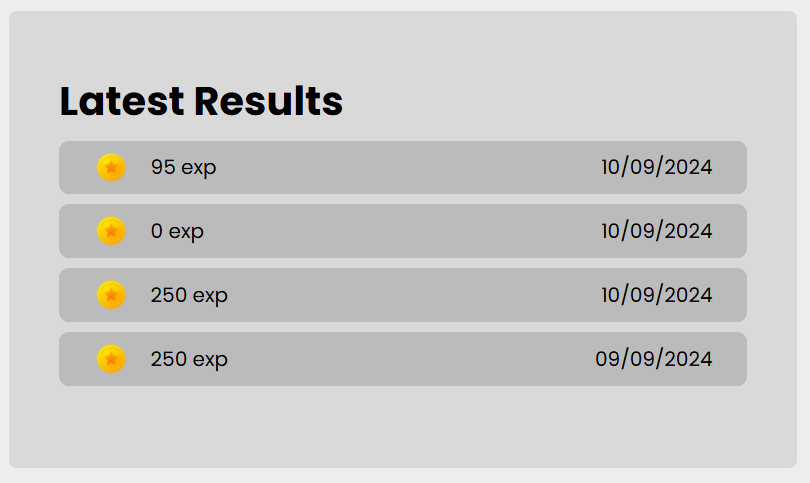
**Personal user details** - The user’s name, level, sign up date, the percentage of users below him in the rankings, and his top language in terms of experience points.

**Achievements** – Tracks the user’s progress in the various achievements available in the website. Each achievement consists of multiple levels, with the requirements being raised for each level.

**Languages** – Tracks the user’s rank in each language he has progressed in out of the available ones. The available ranks for each language are novice, advanced, and master. The rank represents the user’s proficiency in the language, and as the user improves his rank, the difficulty of the given exercises is raised as well.



Hovering over a language list item displays the amount of experience points acquired for that specific language, and the amout required to get to the next rank.



**Latest Results** – displays the last several results of lessons or chat sessions done by the user, and their respective dates.

## About

The About screen displays general information about GPTalk’s features, along with contact information.

# Maintenance Guide

The following section contains instructions for setting up the application for further maintainence and updates.

**Initial Setup:**

* Install Node.js
* Clone the GPTalk repository

**Backend:**

* Provide email and its password for an email used by nodemailer (to send registration notifications). Place the values in the .env file in the fields ADMIN\_EMAIL and ADMIN\_PASSWORD.
* Provide an OpenAI API key to send requests to the API. Place the value in the .env file in the field OPENAI\_API\_KEY.
* Provide a URL for the MongoDB database. Place the value in the .env file in the field DB\_URL.
* MongoDB Compass is recommanded to view the gptalk database contents if it is deployed locally, otherwise MongoDB Atlas can also be used.
* Install backend dependencies with *npm install*.
* Build the project with *npm run build*, start it with *npm start*.

**Frontend:**

* Install frontend dependencies with *npm install*.
* Build the project with *npm run build*, start it with *npm start*.

For more detailed information on libraries used by the application see Readme.md files located in the roots of gptalk\_backend and gptalk\_frontend folders.

# 

# References

Angular. (n.d.). *Angular Docs*. Retrieved from https://angular.dev/

Express.js. (n.d.). *Express.js Docs*. Retrieved from https://expressjs.com/

Inc., M. (n.d.). *MongoDB Docs*. Retrieved from https://www.mongodb.com/

Jest. (n.d.). *Jest Docs*. Retrieved from https://jestjs.io/

OpenAI. (n.d.). *OpenAI API Reference*. Retrieved from https://platform.openai.com/docs/api-reference

PrimeFlex. (n.d.). *PrimeFlex Docs*. Retrieved from https://primeflex.org/

PrimeNG. (n.d.). *PrimeNG Docs*. Retrieved from https://primeng.org/