

**Bilkent University  
Department of Computer Engineering**

**Senior Design Project  
*T2314  
Linguist AI***

**Detailed Design Report**

***22001943, Yağız Can Aslan,*** [***can.aslan@ug.bilkent.edu.tr***](mailto:can.aslan@ug.bilkent.edu.tr)

***22003017, Kardelen Ceren,*** [***kardelen.ceren@ug.bilkent.edu.tr***](mailto:kardelen.ceren@ug.bilkent.edu.tr)

***22002811, Selim Can Güler,*** [***selim.guler@ug.bilkent.edu.tr***](mailto:selim.guler@ug.bilkent.edu.tr)

***22002223, İlkim Elif Kervan,*** [***elif.kervan@ug.bilkent.edu.tr***](mailto:elif.kervan@ug.bilkent.edu.tr)

***22003850, Tolga Özgün,*** [***tolga.ozgun@ug.bilkent.edu.tr***](mailto:tolga.ozgun@ug.bilkent.edu.tr)

***Supervisor: Halil Altay Güvenir***

***Course Instructors***

***Atakan Erdem***

***Mert Bıçakçı***

**15.03.2024**

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

**Table of Contents**

**1 Introduction 3**

[1.1 Purpose of the system 4](#_6z4prtgtmyhi)

[1.2 Design goals 4](#_olyjyood1otw)

[1.2.1 Usability 4](#_vifjqymnoxrg)

[1.2.2 Reliability 5](#_hnlbqefrp23s)

[1.2.3 Performance 5](#_v9zfquh5ctv1)

[1.2.4 Supportability 6](#_c2cch7x7hkb)

[1.2.5 Scalability 6](#_bolj3vmzyd10)

[1.2.6 Security & Privacy 6](#_foamz6aiee10)

[1.3 Definitions, acronyms, and abbreviations 7](#_46tnqivqiy86)

[1.4 Overview 7](#_6txx29at3fuz)

[**2 Current software architecture 8**](#_381dl5jw1pm1)

[**3 Proposed software architecture 9**](#_rhprmpvxx5s6)

[3.1 Overview 9](#_n1fbmg103s7i)

[3.2 Subsystem decomposition 9](#_8w487f8xnbzf)

[3.3 Hardware/software mapping 10](#_2ljgxkgz27gk)

[3.3.1 Mobile Application 10](#_fsxbgn6g27n2)

[3.3.1.1 Microphone and Speaker Access 10](#_bivrx0gqtlob)

[3.3.1.2 Touchscreen Support 10](#_t76kbh5cjrh8)

[3.3.1.3 Notifications Support 11](#_xq5030uytoz9)

[3.3.1.4 Image Gallery - Filesystem Access 11](#_6iq16lys122w)

[3.3.2 Machine Learning Subservice 11](#_hm0ccrxmvemh)

[3.4 Persistent data management 11](#_lzr1r0f0ckbo)

[3.5 Access control and security 12](#_xqgep76kur4s)

[3.5.1 Authentication and session management 12](#_ht4zmbgkjcgi)

[3.5.2 Layered authentication checks 12](#_wr7pyawyka5y)

[**4 Subsystem services 13**](#_q345v6x4bj1s)

[4.1 Client Subsystem 13](#_x5jzal3wxjq8)

[4.2 Backend Subsystem 14](#_bdqipm3wkuhd)

[**5 Test Cases 17**](#_qahzce55kpbw)

[**6 Consideration of Various Factors in Engineering Design 35**](#_62vj6h906brr)

[6.1 Constraints 35](#_jh4mkjow2pld)

[6.2 Standards 37](#_752x6nf7ua6l)

[6.3 Risks and Alternatives 38](#_abjait2teln4)

[**7 Teamwork Details 40**](#_ge0crfjj40cu)

[7.1 Contributing and functioning effectively on the team 40](#_nh3ojs98a7bv)

[7.2 Helping creating a collaborative and inclusive environment 49](#_71ktkjxal8cg)

[7.3 Taking lead role and sharing leadership on the team 49](#_owrv9gqh8af6)

[**8 References 50**](#_n7fb8vz8vcq1)

Detailed Design Report

*T2314: Linguist AI*

# Introduction

English is one of the most spoken languages in the world, with millions of native and non-native speakers [1]. It has become the default language of international events, businesses, technology, etc. Therefore, many people aim to learn English. For instance, in the Turkish education system, students are taught English as a second language for ten years starting from elementary school. That said, most students cannot speak English properly due to the ineffective traditional teaching methods, including grammar and vocabulary memorization using textbooks and flashcards. Students cannot incorporate the language into their everyday lives using the aforementioned methods and, therefore, forget the memorized rules of the language. Also, there is a lack of resources when it comes to writing and speaking materials, as the main focus of language teaching is based on grammar and vocabulary. However, language is not something to be memorized, it is a culture that should be experienced.

In addition to the traditional language learning techniques, there are applications that students can use to learn English, like Cambly and Duolingo. Duolingo is a mobile application that offers lessons for various skill levels [2]. These lessons aim to teach grammar and vocabulary using different activities such as filling in the blanks, translating sentences, speaking, and listening. On the other hand, Cambly is an online platform designed to connect students with native speakers [3]. While using Cambly, students have a chance to interact with a native speaker during the classes. However, the lessons in Duolingo are not customized to individuals and might get boring after a while. Also, they are designed for beginner and intermediate learners. Therefore, someone who wants to speak more advanced or learn academic vocabulary cannot use Duolingo. As for Cambly, it allows students to experience the language culture. However, it might not be affordable for all students. Also, students with self-esteem problems and shyness would be uncomfortable using this app.

LinguistAI is an online mobile application where users can learn vocabulary and practice their language skills using a chatbot. Our application detects the skill level of the user during the chat sessions and adapts to the user’s level. Also, the application determines if the user is correctly using a vocabulary to make sure the user is actively learning. Moreover, it personalizes the conversations based on a user model that includes the user's interests, demographic information, and more. Users can have a conversation with the bot about a topic or scenario they like such as buying a coffee or plane ticket. Our application will include gamification elements such as achievements, leaderboards, and daily streaks. The main focus of Linguist AI is to teach words to users by repeatedly using unknown words for a user in different contexts and measuring the level of knowledge of the user for each word. All in all, Linguist AI will enable users to learn English tailored to their level by eliminating artificial learning items such as flashcards and textbooks and offering a sincere conversational environment.

The rest of this report will describe the purpose of the system and its design goals, describe the current software architecture and explain the proposed software architecture, showcase the subsystem services, and list a set of test cases. The report will end with consideration of various factors in engineering design and describing teamwork details.

## Purpose of the system

One of the primary goals of Linguist AI is to make it easier to learn and practice English. For this reason, our app addresses the shortcomings of traditional language learning techniques, especially those that rely on rote memorization, as well as the one-size-fits-all strategy they frequently use, which is unable to take into account different learning styles and needs, and the little opportunities they provide for practice in real-world situations. Instead, it provides an interactive, customized experience that adjusts to each user's proficiency and interests, aiming to make learning more pleasurable and productive.

Furthermore, LinguistAI seeks to increase confidence and competence in writing, listening, speaking, and vocabulary usage in English. Practicing writing or speaking to an AI motivates and increases the confidence of learners as they do not fear making mistakes as they would with a human [4, 5]. Studies show that practicing with a (voiced) chatbot increases speaking skills and vocabulary retention significantly [6, 7, 8].

In a similar vein, Linguist AI encourages vocabulary learning through more organic means as opposed to depending only on memorization-based techniques. For example, it exposes users to words marked as previously unknown in conversations more frequently, allowing them to see how they are used "in the wild" and in various contexts. The words are then asked in multiple-choice questions to reinforce the learning. This method's aim is to help users retain more vocabulary while also gaining a deeper understanding of how words are used in everyday contexts.

LinguistAI seeks to boost user motivation and engagement by implementing gamification features like leaderboards and achievements. In addition to being used to make learning English more enjoyable, this strategy encourages users to actively engage in the process of acquiring the language. Additionally, by providing a language practice companion accessible from any geographic location and at a fraction of the cost of private tutoring, LinguistAI makes language practice available to a diverse range of learners.

## Design goals

This section expands upon the design goals of Linguist AI under Usability, Reliability, Performance, Supportability, Scalability and Security & Privacy sub-sections.

### Usability

Linguist AI will be a mobile application. Since Linguist AI will be a mobile application, the users will be able to use our application anywhere which can include while commuting on the bus, when waiting in line, while waiting for food, and many more examples. As such, it is vital that the application is accessible and easy to use. The users must be able to utilize our application without struggle. Therefore, the application must have an intuitive and simple user interface, ensure accessibility for all types of users (different types of color blind etc.), have a responsive design, maintain optimal performance, include methods for gathering user feedback and error reporting, be compatible across a variety of platforms and maintain a consistent design. The application also should have an easy-to-follow and informative user onboarding procedure upon first use, to maintain user retention and make the application easier to understand and use. In order to adhere to the aforementioned usability guidelines for our project, the users must be able to reach all parts of the application with at most 5 clicks. For ease of use, the navigation bar should be visible on all screens, and the back buttons must always be in the same place (for instance, top left). We have defined the following constraints to measure usability:

* Each main subsystem, such as chat and leaderboard, should be accessible within 2 clicks for every logged-in user.
* Each login step should take 4 steps at maximum with valid credentials.
* All back buttons should be placed on the top left of the screen, with 16 eDPI for left and top margins.
* Each main subsystem should have a visible bottom navigation.
* All network requests should be asynchronous so that users can interact with the front end while the backend response is being generated.
* All in-progress network requests should be clearly indicated to the user with a loading visual placed as the closest element to the target component.

### Reliability

In order for our application to achieve its goals, the application must provide a consistent experience. All systems of our application (backend servers, databases, LLM integration, mobile application etc.) must cooperate smoothly without major errors. To provide a smooth experience to our users, all of our systems should have a 95% uptime. The user must be able to continuously use the application without encountering a major error in any step. Moreover, the responses of our conversational AI models must not be strange or unsettling, the users must know that whenever they use our application, they will be subject to the same experience.

### Performance

As stated in section 1.2.1, our users can use our application almost anywhere. As such, it is important for our application to be responsive. For this reason, the performance of the application will be measured through various metrics such as time to load (TTL), mean time to respond (MTTR) and time to first byte (TTFB). An upper bound is declared for all of these metrics in various contexts. The constraints considered for such metrics are as follows:

* The time to the first byte of the chat system should be strictly under 10 seconds. This also applies to a potential version of the chat which uses text streaming via sockets.
* Mean response time of the authentication service should be less than 5 seconds.
* Average time to load a page on the mobile application should be less than 3 seconds.
* Adding new words to the word list should take less than 2 seconds.
* Loading user profiles should take less than 4 seconds.
* Mean response time of the main subsystems besides chat and authentication should be less than 4 seconds.

If these conditions are not met, the users will have to wait a considerable amount of time to utilize our features which will decrease our application’s usage and language teaching effectiveness significantly. With these metrics in mind, our expectation is having a UI that is performant and responsive, so the application does not feel clunky, slow or old, and the connections between the subsystems of our project such as backend servers, databases, LLM integration and mobile application happening relatively fast to enable the aforementioned smooth and responsive experiences.

### Supportability

In order to reach a broader user base, our mobile application must be available for a variety of platforms, including Android and iOS. As such, we must maintain the application for different operating systems and mobile devices. Our application must be able to be maintained and expanded upon easily. This can be addressed by following good software development practices and adhering to tested and established design paradigms such as Object Oriented Programming. In order to adhere to these values, we established the following constraints:

* The minimum Android SDK version should be 24.0 which encompasses more than 98% of the active Android user base, and is the recommended minimum by Google.
* The minimum iOS version should be 15 which is the version recommended by Apple, as more than 94% of users are on iOS 15 and later versions.

### Scalability

Since our application is a mobile application, it will be easy for new users to download and start using our application. Therefore, our systems (backend servers, databases, LLM integration, mobile application etc.) must be prepared and ready to handle a surge in users and server loads. Our system must be able to handle at least 1000 concurrent users. We must also develop our systems in such a manner that migration, adding new features or distributing our application to new vendors (for mobile applications, like App Store, Google Play Store etc.) will not cause significant problems.

### Security & Privacy

Linguist AI will store user data to provide a specialized learning experience. The stored data may include names, interests, passwords, emails, conversation history and many more types of information. Henceforth, it is crucial that our systems are as secure as possible to protect sensitive data and that we handle such data with extreme care. To achieve this, we can utilize complex encryption algorithms and various other tried and tested security protocols. Moreover, we will follow GDPR (General Data Protection Regulation) and KVKK (Kişisel Verilerin Korunması Kanunu) guidelines closely and will seek user consent whenever necessary, to strike a balance between personalization and privacy. We are implementing the following procedures for our security purposes:

* All passwords must be hashed with SHA-256 and salted.
* All sessions must be given a session token by JWT and all requests should have a valid access token.
* All users should be provided with a refresh token to regenerate the access token when expired.
* Access token should be valid for 24 hours.
* Refresh token should be valid for 168 hours.

## Definitions, acronyms, and abbreviations

XP: Experience Point,

JSON: JavaScript Object Notation,

JWT: JSON Web Token,

TTL: Time to Load,

MTTR: Mean Time to Respond,

TTFB: Time to First Byte,

ML: Machine Learning,

LLM: Large Language Model,

GDPR: General Data Protection Regulation,

KVKK: Kişisel Verilerin Korunması Kanunu,

CPU: Central Processing Unit,

GPU: Graphical Processing Unit

## Overview

For those learning English as a foreign language, LinguistAI is a mobile application that allows for text- or speech-based interactions with an LLM-based chatbot to help with vocabulary, writing, speaking, and listening practice. To ensure meaningful conversations, it is designed for users with at least an A2 proficiency level in English.

Users have the option to select a chatbot based on their proficiency level and interests. Users can simply click on the word when they encounter unfamiliar words during conversations to view its definition and example sentences. Additionally, users can save unknown words to a list or manually add words to custom lists. The chatbot then incorporates these words into future conversations, presenting them in diverse contexts to help users become familiar with them. In addition, users can converse verbally with the chatbot to improve their listening and speaking abilities.

In order to determine how much the user learned from interacting with the AI, and to reinforce learning, vocabulary is tested using multiple-choice questions at the beginning and end of the chatbot conversation. An LLM examines how users use different vocabulary in their sentences so that each word's usage can be scored depending on its context. The questions and word scores are used to assess the user's familiarity with the words on a scale of five confidence levels.

Beyond these functionalities, LinguistAI incorporates gamification components such as login/chat streaks, experience points, statistics about the number of messages sent and words learned, chatbots with different personalities, leaderboards, a friendship system, achievements, and daily quests. The purpose of these features is to boost users' motivation for learning languages. The chatbot also builds a user profile during chats by adding preferences, demographic information, and topic suggestions. This user profile is then taken into consideration when creating LLM responses for a fun and engaging learning experience.

# Current software architecture

Language-learning applications such as Duolingo, Memrise, and Mondly can be considered competitors of Linguist AI in the e-learning space, especially considering that they were among the first to introduce AI-driven chat features for language learning. Although it is difficult to obtain comprehensive information about the software architecture of these companies, we can focus on Duolingo's architecture because they are the most transparent about sharing details concerning their system.

Duolingo's architectural approach focuses on addressing scalability concerns and improving code maintainability through the use of modern architectural patterns such as Model-View-ViewModel. The Android development team employed Hilt and Dagger, two components of Android Jetpack, to implement repository patterns. This decision allows for a clearer separation of concerns within the codebase, resulting in more efficient development and debugging processes. Hilt's reduction in boilerplate code that needs to be written and Dagger's ability to produce readable, understandable code with verbose error logging additionally enhance developer productivity and code quality [9].

Additionally, Duolingo redesigned its entire system in 2017 in order to switch from Python to Scala. The need to resolve scalability issues and performance bottlenecks in its original Python-based infrastructure drove this change. The adoption of Scala, a functional programming language with statically typed syntax, was motivated by its superior performance capabilities and adaptability to complex systems, which matched Duolingo's functional requirements. This move improved the user experience overall by guaranteeing reportedly 100% uptime for Duolingo's essential components and reducing latency significantly [10].

Lastly, Duolingo's infrastructure stack comprises Finatra as the HTTP server, along with Guice for dependency injection and Mockito for testing. Leveraging AWS Elastic Beanstalk for deployment provided automated handling of rolling deployment, load balancing, autoscaling, and monitoring, ensuring the robustness and scalability of the platform.

# 

# Proposed software architecture

## Overview

Linguist AI’s architecture can be divided into two main parts: client-side architecture and server-side architecture.

The client-side architecture consists of developments made on the mobile application that will utilize the functionalities offered by the server side. Server-side architecture is built based on microservice architecture and currently has three microservices: user service, machine learning (ML) service, and dictionary service. The client-side communicates with these microservices via a custom-built API gateway. Each of the microservices works independently of each other except for communication necessary for some tasks.

For the software, users are required to use their mobile devices since our software is designed and developed for mobile. Our servers are deployed on the same virtual server. In the future, we aim to move some of our services to AWS.

Our application requires persistent data management for use cases like user information, keeping track of conversations and messages sent for each conversation, friends of a user, achievements for a user, etc. For that purpose, we use database systems deployed in our servers which are accessed by our services such as ML service and User service.

Since we have users registering for our application, we also care about their data safety. Registering to the application is required in order to access all of its features. Once a user is registered they can access the features of the application. We use JWT tokens with server sessions to secure each user’s account. We also do not allow any unauthenticated “user” to send requests to our servers. We check for this both on the client and server side for additional security.

## Subsystem decomposition

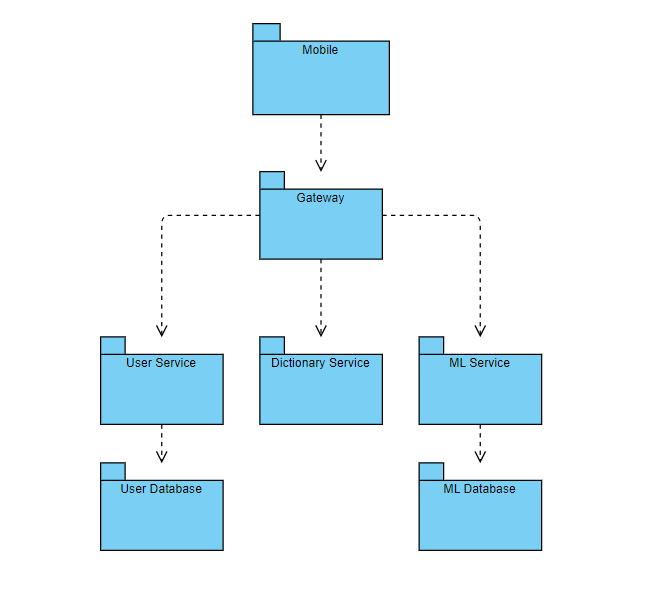


Figure 1.

Linguist AI is composed of mobile, backend, and database components, all of which are independent of each other. This independence, allows us to use different languages and frameworks according to our needs. Furthermore, if we decide that our current technology stack does not meet our requirements, we can quickly switch to another framework for that service without interrupting the implementation of other components.

In the mobile component user interface and its business logic are implemented. This is the only part where end users can interact with Linguist AI. The mobile component sends and receives requests from the gateway, allowing us to control common logic, such as authentication, from a top level. Therefore, we don’t need to implement the authentication in each backend service.

Account-related logic, such as login and register, word lists, and gamification elements are handled in the user service. In this service, controller, service, and repository layers are implemented for each of the main logical groups. Furthermore, user service uses a database for storing user, word bank, and gamification related data.

The primary purpose of the dictionary service is to send requests to external APIs for fetching the meanings, pronunciation, and example usages of the requested vocabulary. This service uses the aforementioned layer pattern as well in order to provide a maintainable and clear structure.

The machine learning (ML) service is used for features such as conversation generation, user profile extraction, and word scoring. ML service stores its data in a separate database.

## Hardware/software mapping

Linguist AI has two main modules that heavily relies on hardware/software mapping. First, we have a mobile application that we distribute to all of our end users. Secondly, we have our machine learning subservice where we utilize GPUs and quantized models on CPUs to run our LLM and ML solutions.

## Mobile Application

Linguist AI offers a mobile application as a user interface, built with a well-used framework, React Native. This framework helps us to abstract the hardware connections within different operating systems, and help us offer solutions to all widely used mobile operating software such as Android and iOS. Our hardware-software mapping considerations for our mobile application is as follows.

## Microphone and Speaker Access

Central to the LinguistAI experience is the ability for users to interact with the chatbot through both speech and text. The application leverages the phone's built-in microphone to capture the user's speech for processing and analysis. Similarly, the speakers are utilized to play back the chatbot's responses, enabling a dynamic speaking and listening practice environment. This hardware integration facilitates a natural conversation flow, crucial for language learning.

## Touchscreen Support

The application is designed for smartphones, with the touchscreen serving multiple purposes. It allows users to navigate the app, select words for definitions, and interact with various elements of the gamification features. The intuitive touch interface supports an engaging and interactive learning process.

## Notifications Support

Utilizing the device's built-in notification system, LinguistAI keeps users engaged and motivated by sending reminders for daily quests, achievements, and other gamification elements. This feature ensures that users maintain a regular interaction pattern with the application, enhancing the learning process.

## Image Gallery - Filesystem Access

For personalization, the application requests access to the phone's image gallery. This feature allows users to upload profile pictures, fostering a more personalized and engaging user experience.

## Machine Learning Subservice

* + - 1. **GPUs for Fine-Tuned LLM:**

These GPUs are high-performance GPUs rented on an hourly basis. They are utilized for deploying the fine-tuned LLM to handle compute-intensive tasks, including real-time user requests and complex computations. Dynamic scaling based on the current load, with additional GPU resources allocated during peak times to maintain performance levels.

* + - 1. **CPUs for Quantized LLM:**

For our quantized LLM models, we utilize standard multi-core CPUs available within the existing server infrastructure. This system supports the quantized version of the LLM for less intensive tasks or when the load is manageable within the CPU's computational capacity. The activation is triggered when the system detects a load level that can be efficiently handled by the CPU, or during low traffic periods to reduce operational costs.

## Persistent data management

Our systems heavily depend on data storage as we are storing the conversations between users and the bot, and further analyzing these data in order to obtain user profiles. To that end, this service uses a PostgreSQL database. Furthermore, in order to display the last chat messages to the user, conversations are also saved in the local storage of user devices. This storage is synchronized every time user conversation data is fetched from the backend.

In order to have an independent, scalable, and maintainable system, we opt for separating ML and user databases. Therefore, the user microservice uses a MySQL database for storing user data, unknown word lists of users, streaks, experience points (XP), friendships and requests, quests, achievements, and statistics about users’s learning process. In order to efficiently maintain database operations without considering the low-level connection details, our system uses JPA (Java Persistence API). This library also provides interfaces to easily handle queries without writing SQL codes.

## Access control and security

We have a multi-layered approach to authentication and session management to achieve the safety of user data and maintain the integrity of our services.

### Authentication and session management

**JSON Web Tokens (JWT):** Upon successful authentication, users are issued both an access token and a refresh token. These tokens, encrypted and securely transmitted, serve as the basis for maintaining user sessions.

**Session Validity:** User sessions (access tokens) expire after 15 minutes of their issue date. If a user continues to use the application after this point, our system utilizes the refresh token to automatically get a new access token. The refresh token mechanism can only be used within a day of users’ last login. Session validity is a really important topic in terms of the security of user sessions. We take this seriously and keep the access token’s lifetime short to overcome security issues.

**Client and Server-Side Management**

While session data (JWT) is securely stored on our servers, the access and refresh tokens are managed on the client side within secure storage mechanisms. This ensures that only authenticated users can access our application and its services. Requests with invalid or expired tokens are rejected by our API gateway, thus preventing unauthorized access attempts. Thereby, only an authenticated user can use our services.

### Layered authentication checks

**Client-Side Controls:** Each page of our application is guarded with specialized authentication controls. Upon accessing any page, the user's authentication status is immediately verified on the client side using the stored access and refresh token information. If authentication is lacking or unverified, the user is redirected to the login page, preventing unauthorized access.

**Server-Side Validation:** Complementing client-side checks, our server infrastructure employs similar authentication validation protocols. Requests originating from unauthenticated sessions are promptly denied. This ensures that each HTTPS request carries the correct authentication information in its header.

# 

# Subsystem services

## Client Subsystem

The client side of the project is a mobile application. The client subsystem consists of pages where each page has its own logic and user interface implementation. Moreover, navigation from a certain page is limited. Meaning that the user can only go to certain pages from the page they are in.

We only have four pages that are accessed without authentication: Landing Page, Login Page, Register Page, and Forgot Password Page. All other pages require an authenticated user to be logged in.

The client system is displayed in Figure 2.

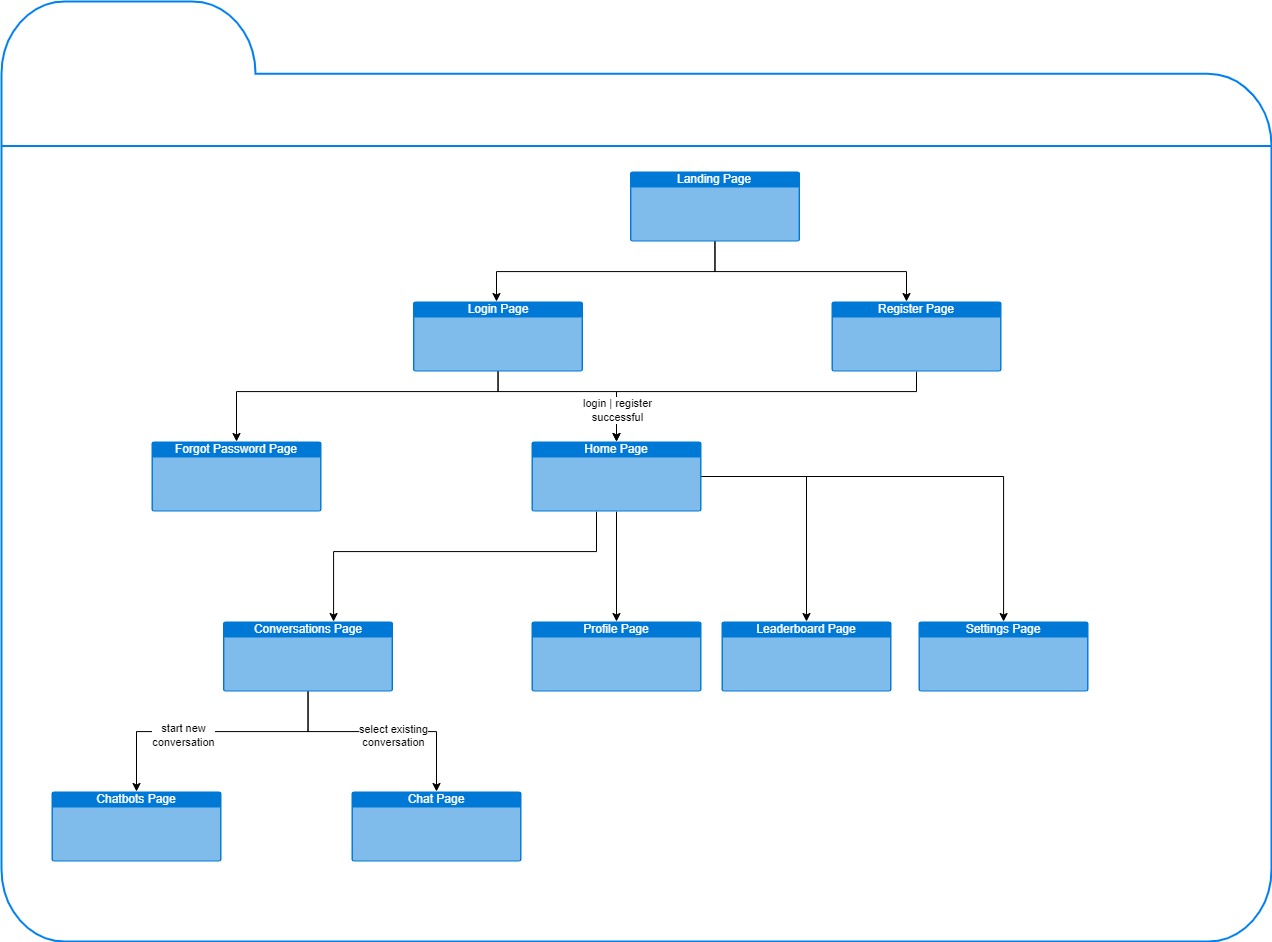


Figure 2.

## Backend Subsystem

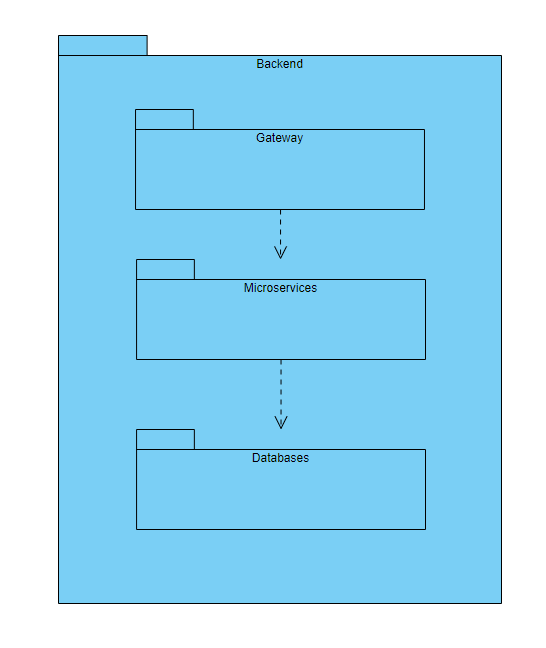


Figure 3.

The general structure of the backend is given above. All requests are arrived at the gateway and then redirected to the designated microservices. Afterwards, database queries are executed to complete the process, and the response is returned to the client side.

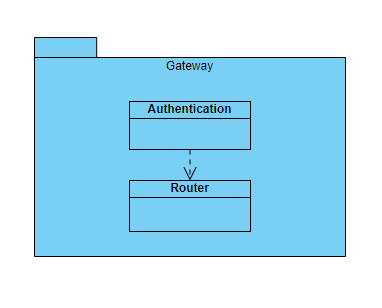


Figure 4.

If the incoming requests require an authentication process, the gateway authenticates users and then routes the requests to the related microservice.

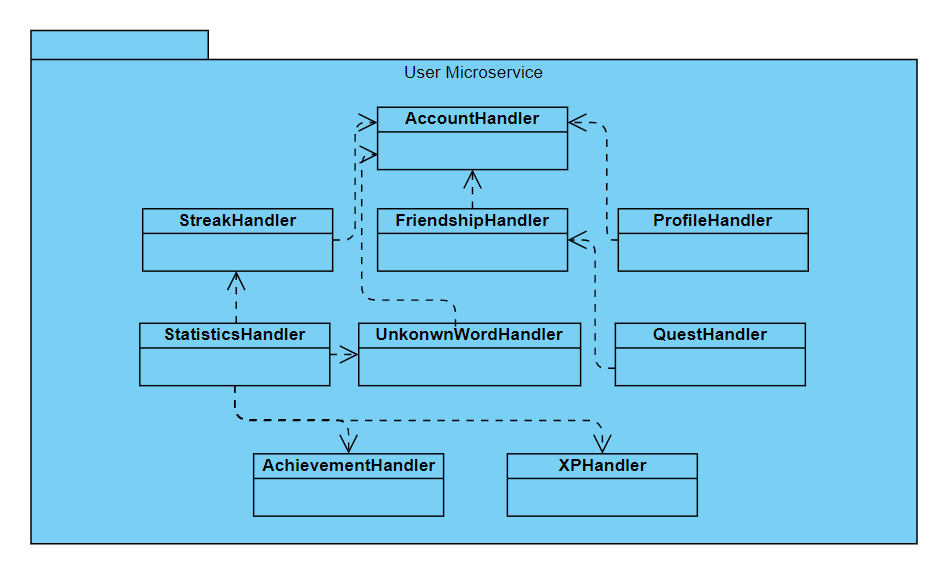


Figure 5.

Above is the diagram of the user microservice and the handlers represent the controller, service, and repository layers altogether. Apart from the login and register endpoints, which are managed by the AccountHandler, all endpoints are authenticated. The arrows imply that the originating handler uses the pointing handler. For example, QuestHandler uses FriendshipHandler as quests are created between friends.

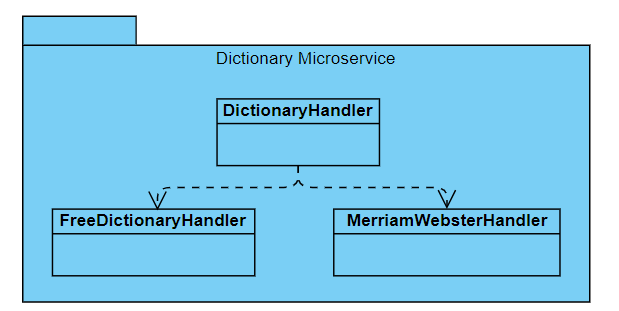


Figure 6.

The main purpose of the dictionary microservice is to retrieve dictionary data based on user requests. There are two resource dictionaries, Merriam Webster and Free Dictionary. The DictionaryHandler chooses between these dictionaries based on the remaining request quota.

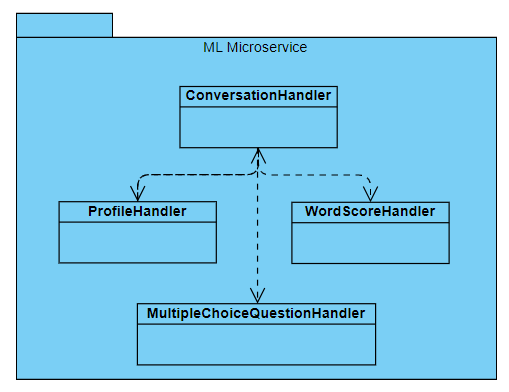


Figure 7.

In this microservice, four main features are managed, as indicated above in the diagram. Based on users' messages, their profiles, including their likes and dislikes are extracted in the ProfileHandler. Then, ConversationHandler is fed back with users’ profiles, so that the bot can talk about topics users like. Furthermore, WordScoreHandler is implemented in order to evaluate users’ performance during messaging. In the MultipleChoiceQuestionHandler, questions are generated and evaluated to test the knowledge of users. All of these handlers make use of a LLM model that is accessible in the microservice.

# 

# 

# Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 1 | Category | Non-Functional | Severity | Critical |
| Objective | Any unauthenticated access attempt to authenticated pages should be redirected to the login page. | | | | |
| Steps | Step 1. Try to access either one of these pages: Home, Conversations, Chat, Profile, Leaderboard, Settings | | | | |
| Expected | Users should not access pages other than landing, login, and register without authentication. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 2 | Category | Non-Functional | Severity | Major |
| Objective | Response from any of the chatbots should not exceed 5 seconds | | | | |
| Steps | Step 1. Login to the application  Step 2. Go to the conversations page  Step 3. Initiate a conversation  Step 4. Write a text message and send  Step 5. Measure the time takes for the bot to respond | | | | |
| Expected | The time to respond should be less than 5 seconds | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 3 | Category | Non-functional | Severity | Minor |
| Objective | Any functionality should be 3 steps away from where the user is at. | | | | |
| Steps | Step 1. Go to any page  Step 2. Go depth-first to any linked page from your current page | | | | |
| Expected | Navigation from a page to a functionality should take at most 3 steps | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 4 | Category | Functional | Severity | Major |
| Objective | User registration fails if password is too weak | | | | |
| Steps | Step 1. Go to the Registration page  Step 2. Attempt to register with a password that is less than the minimum length requirement.  Step 3. Attempt to register with a password that does not include at least one uppercase letter.  Step 4. Attempt to register with a password that does not include at least one lowercase letter.  Step 5. Attempt to register with a password that does not include at least one digit.  Step 6. Attempt to register with a password that does not include at least one special character. | | | | |
| Expected | All attempts in steps 2-6 should not register a new user. Instead, a relevant fail notification should appear on the screen. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 5 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the users can log in with the correct credentials and are redirected to the home page | | | | |
| Steps | Step 1. Login with an existing user’s credentials | | | | |
| Expected | Authentication succeeds, and user accesses the Home page. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 6 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the user receives a response from the chatbot when they send a message | | | | |
| Steps | Step 1. Send a message to a chatbot  Step 2. Wait for the response from the chatbot and observe the output | | | | |
| Expected | The chatbot should respond after a few seconds. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 7 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the user cannot send any messages while waiting for the response of their previous message | | | | |
| Steps | Step 1. Send a message to a chatbot  Step 2. While waiting for bot’s response try to send another message | | | | |
| Expected | Send button should be disabled and the user should not be able to send multiple messages at the same time. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 8 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the user cannot create multiple word lists with the same name | | | | |
| Steps | Step 1. Go to the word lists page  Step 2. Create a word list with some name  Step 3. Try to create another word list with the same name | | | | |
| Expected | New word list should not be created and user should be notified that word list names should be unique | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 9 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that when a user clicks on a word on the chat screen, their meaning pops up on the screen | | | | |
| Steps | Step 1. Go to a chat  Step 2. If there are no messages send one  Step 3. Tap on a “meaningful” word | | | | |
| Expected | The meaning of the word with examples should pop up on a card. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 10 | Category | Functional | Severity | Minor |
| Objective | This test case is to verify that the login streak is only displayed on the first login of the day | | | | |
| Steps | Step 1. Login to the application  Step 2. Sign out and log in again | | | | |
| Expected | The login streak should be displayed on the first try, but not on the second login. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 11 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the system is durable against SQL injections. | | | | |
| Steps | Step 1. User opens the app and tries to register or login.  Step 2. User enters some malicious SQL code in some of the input fields, such as username. | | | | |
| Expected | The system does not crash and sends an error response to user | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 12 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the email field only accepts strings in proper email format. | | | | |
| Steps | Step 1. User tries to sign up to the system.  Step 2. User enters invalid email. | | | | |
| Expected | System displays an error message for the user to update the email field correctly. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 13 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the authenticated endpoints can be called if a valid token is added to the request header. | | | | |
| Steps | Step 1. User sends a request to an authenticated endpoint, for example, auth/test endpoint, without providing a token. | | | | |
| Expected | An error response saying that the user is not authorized is returned. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 14 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that an email can be used for only one user. | | | | |
| Steps | Step 1. User opens the app and tries to sign up.  Step 2. User signs up with an email.  Step 3. User logs out and clicks on the sign up button again.  Step 4. User enters the same email they used in Step 2. | | | | |
| Expected | System returns an error response indicating that the user is already registered with this email. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 15 | Category | Functional | Severity | Minor |
| Objective | This test case is to verify that the authentication token is valid for 1 day. | | | | |
| Steps | Step 1. User logs in using the login endpoint.  Step 2. User tests authentication token using auth/test endpoint.  Step 3. After 24 hours user sends test request again. | | | | |
| Expected | System returns an error message indicating that the user is not authenticated. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 16 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the dictionary service can return data even if the Free Dictionary quota is exceeded.  Note: Linguist AI uses two different dictionaries with different quotas. Our main dictionary is Free Dictionary and it has 400 request quotas per minute. | | | | |
| Steps | Step 1. User opens the word list tab.  Step 2. User creates a word list by clicking on the plus button.  Step 3. User adds at least 50 words to their word list.  Step 4. User clicks on the home tab and then opens this word list.  Step 5. User repeats Step 4, 9 times in a minute. | | | | |
| Expected | User can see the meanings of the vocabulary in the word list every time they open the word list. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 17 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the system can return an error message when one of the microservices is down. | | | | |
| Steps | Step 1. For test purposes, dictionary service container is stopped.  Step 2. User logs in to the system successfully.  Step 3. User clicks on the word list tab. | | | | |
| Expected | System returns an error message indicating that the requested server is currently down. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 18 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the when user sends a friend request to another user, that user will be notified. | | | | |
| Steps | Step 1. User logs in to the system as different users on different devices.  Step 2. User opens the friends tab and searches for the username logged in on the other device.  Step 3. User sends a friend request to the user found in Step 2. | | | | |
| Expected | User sees a notification message indicating that there is an incoming friend request on the other device. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 19 | Category | Functional | Severity | Minor |
| Objective | This test case is to verify that the user can only send friendship requests to other users they are not already friends with. | | | | |
| Steps | Step 1. User opens the app and, in the friends tab searches for a user.  Step 2. User sends a friend request to another user.  Step 3. User clicks on the home tab and then returns to the friend tab and tries to send a friend request to the same user. | | | | |
| Expected | User cannot send friendship request again as there is no button for it. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 20 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the user can be friends with other users after accepting their friend request. | | | | |
| Steps | Step 1. After Test 18 is completed, user opens the app and views the friend request.  Step 2. User accepts the request by clicking the accept button.  Step 3. User clicks on the friends tab. | | | | |
| Expected | User sees their new friend in the friends tab. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 21 | Category | Non-Functional | Severity | Critical |
| Objective | This test case is to verify that the application securely stores passwords by hashing them. | | | | |
| Steps | Step 1. Register a new user  Step 2. Access the database where passwords are stored  Step 3. Retrieve the stored password associated with the registered user | | | | |
| Expected | The password stored in the database should not be in plaintext, instead it should be hashed using the secure hashing algorithm SHA-256. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 22 | Category | Non-Functional | Severity | Critical |
| Objective | This test case is to verify that the ML subsystem (LLMs) can scale up to handle 100 concurrent users. | | | | |
| Steps | Step 1. Calculate the amount of load required for each ML subsystem endpoint.  Step 2. Simulate a load for each endpoint.  Step 3. Observe the time taken for each response, issues and system performance. | | | | |
| Expected | Response times being within the acceptable threshold as defined in the performance requirements and no request should fail. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 23 | Category | Non-Functional | Severity | Major |
| Objective | This test case is to verify automated build and deployment processes for User Service via CI/CD pipeline. | | | | |
| Steps | Step 1. Trigger a build through commit or pull request  Step 2. Observe the Github Actions logs  Step 3. Verify the deployment in the testing environment | | | | |
| Expected | The build and deployment processes should complete successfully without errors. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 24 | Category | Non-Functional | Severity | Critical |
| Objective | This test case is to verify that API Gateway correctly routes requests to the corresponding services without any latency spikes. | | | | |
| Steps | Step 1. Create a list for each possible request to each service  Step 2. Send requests to different services via the API Gateway.  Step 3. Measure the latency of each request.  Step 4. Compare with baseline latency values. | | | | |
| Expected | The latency should not significantly increase, routing should be correct and performance should stay in the aforementioned requirements range. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 25 | Category | Non-Functional | Severity | Critical |
| Objective | This test case is to verify that database backups occur as scheduled without impacting service availability. | | | | |
| Steps | Step 1. Observe the scheduled backup process.  Step 2. Create various metrics such as checksum to compare versions before and after backup.  Step 3. Compare the versions to ensure data integrity.  Step 4. Check for any service downtime during the backup. | | | | |
| Expected | Backups should be complete and accurate, with no service downtime. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 26 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the chat feature can handle simultaneous conversations with multiple users without mixing up contexts. | | | | |
| Steps | Step 1. Initiate multiple chat sessions with the LLM simultaneously.  Step 2. Give multiple very unrelated distinct topics for each session.  Step 3. Assess if the conversations remain contextually accurate.  Step 4. Make sure the accuracy with follow-up questions about other chat contexts. | | | | |
| Expected | The LLM should maintain separate and contextually accurate conversations with each user. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 27 | Category | Functional | Severity | Minor |
| Objective | This test case verifies that the statistics displayed in the LinguistAI application accurately represent the number of messages sent by time and by each chatbot. | | | | |
| Steps | Step 1. Open the application and navigate to the statistics section.  Step 2. Note down the current count of messages sent today and by each chatbot.  Step 3. Engage in conversations with different chatbots, sending varying numbers of messages.  Step 4. Repeat step 2. | | | | |
| Expected | The statistics section should display an updated count of messages for the correct date and the correct chatbot. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 28 | Category | Functional | Severity | Minor |
| Objective | This test case verifies that the statistics in the LinguistAI application accurately represent the number of words learned over time. | | | | |
| Steps | Step 1. Open the application and navigate to the statistics section.  Step 2. Note down the current count of words learned today.  Step 3. Navigate to the chat screen, start conversing with the chatbot.  Step 4. Activate a word list with one unknown word.  Step 5. Repeatedly use the word in correct sentences.  Step 6. Answer multiple choice questions about the word correctly.  Step 7. Repeat steps 6 and 7 until the word is shown as mastered in the word list screen (confidence level highest).  Step 8. Navigate to the statistics section and repeat step 2. | | | | |
| Expected | The statistics section should display an updated count of words learned. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 29 | Category | Non-Functional | Severity | Minor |
| Objective | This test case is to verify that the Docker containers auto-restart upon failure. | | | | |
| Steps | Step 1. Force a failure in each service container separately.  Step 2. Observe if each container restarts automatically. | | | | |
| Expected | The container should restart without manual intervention. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 30 | Category | Non-Functional | Severity | Critical |
| Objective | This test case is to verify that the monitoring system triggers alerts for service outages. | | | | |
| Steps | Step 1. Simulate an outage by taking a service offline.  Step 2. Check if the monitoring system detects the outage.  Step 3. Verify that the alerting system sends out a notification promptly. | | | | |
| Expected | The monitoring system should detect the outage, and the alerting system should notify the relevant personnel within 2 minutes. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 31 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that a relevant user profile is generated by the LLM. | | | | |
| Steps | Step 1. User opens the LinguistAI app and starts a conversation with the chatbot.  Step 2. User sends a text saying they like a particular topic.  Step 3. User sends ten more texts. | | | | |
| Expected | The user profile is updated with the information that the user likes the given topic. (Step 3 is to ensure that enough messages are sent to update the user profile since the updates occur in text batches to reduce overhead) | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 32 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that users can effectively engage in verbal interaction with the chatbot. | | | | |
| Steps | Step 1. The user opens the LinguistAI app and initiates a conversation with the chatbot.  Step 2. The user says a sentence in English.  Step 3. The chatbot responds appropriately to the user's spoken input.  Step 4. The user listens to the chatbot's response. | | | | |
| Expected | The chatbot correctly recognizes the user's spoken input, delivering responses that can be listened to with clear and accurate pronunciation. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 33 | Category | Functional | Severity | Major |
| Objective | This test case is to verify the creation of meaningful multiple-choice questions by the LLM. | | | | |
| Steps | Step 1. The user starts a conversation with the chatbot.  Step 2. LLM generates multiple-choice questions based on the active word list. | | | | |
| Expected | The generated questions are relevant and the correct answer option for each question corresponds to the target word. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 34 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the app can assess the user's responses to multiple-choice questions and update the word's level based on the accuracy of responses. | | | | |
| Steps | Step 1. The user starts a conversation with the chatbot.  Step 2. LLM generates multiple-choice questions based on the active word list.  Step 3. The questions are presented to the user.  Step 4. The user answers based on the given options. | | | | |
| Expected | The app accurately assesses the user's responses to multiple-choice questions, and the word's level is updated based on the accuracy of the user's responses. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 35 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that a user can request a password reset and an email containing a reset code is sent to the user's email address, allowing them to change their password. | | | | |
| Steps | Step 1. The user opens the app and clicks on the "Forgot Password?" button on the login screen.  Step 2. The user enters their registered email address and submits the request.  Step 3. The system generates a unique reset code and sends an email containing the code to the user's email address.  Step 4. The user checks their email inbox and retrieves the reset code.  Step 5. The user returns to the app and enters the reset code received via email.  Step 6. The user sets a new password for their account and submits the change.  Step 7. The user attempts to log in with the new password. | | | | |
| Expected | User receives an email containing the reset code. After inputting the reset code, the user is able to set a new password without encountering any errors. The user can log in using the new password after resetting it. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 36 | Category | Functional | Severity | Minor |
| Objective | This test case is to verify that the chatbot responds to non-English or random characters in English. | | | | |
| Steps | Step 1. The user opens the app and starts a conversation with the chatbot.  Step 2. The user enters text in a language other than English or random characters.  Step 3. Repeat step 2 for five different languages. | | | | |
| Expected | Chatbot responds in English, with a text saying it does not understand the message. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 37 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the chatbot responds appropriately when the user inputs hate speech or sexual content. | | | | |
| Steps | Step 1. The user opens the app and initiates a conversation with the chatbot.  Step 2. The user deliberately inputs hate speech, sexual content, or harassing messages into the chat. | | | | |
| Expected | The chatbot recognizes and addresses the inappropriate content by responding with a message that discourages such behavior and promotes respectful interaction. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 38 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the words seen in a conversation can be added to an unknown word list. | | | | |
| Steps | Step 1. The user opens the app and starts a conversation.  Step 2. The user taps on a word.  Step 3. The user adds this word to a word list.  Step 4. The user navigates to the word list screen.  Step 5. The user taps on the relevant word list. | | | | |
| Expected | The word list displays the word added from the conversation screen. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 39 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the words in the activated word list are used in the messages sent by the chatbot. | | | | |
| Steps | Step 1. The user opens the app and navigates to the word list screen.  Step 2. The user activates a word list with at least five words in it.  Step 3. The user navigates to the conversation screen and starts a conversation.  Step 4. The user continues the conversation until the chatbot sends ten messages total. | | | | |
| Expected | At least one word from the activated word list is used in the chat. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 40 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the LLM assigns a score to the unknown words used in a user’s sentence, which is used to update the word’s confidence level. | | | | |
| Steps | Step 1. The user opens the app and starts a conversation.  Step 2. The user sends a text using an unknown (low confidence level) word from the activated word list in a correct sentence. | | | | |
| Expected | Word’s confidence level is updated. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 41 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the user can activate their unknown words list. | | | | |
| Steps | Step 1. User opens the word list screen.  Step 2. User clicks on a pre-existing word list.  Step 3. User clicks on the button to activate the list. | | | | |
| Expected | The newly activated word list shows as activated on the UI and the database. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 42 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that the user can deactivate their unknown words list. | | | | |
| Steps | Step 1. User opens the word list screen.  Step 2. User clicks on a pre-existing word list.  Step 3. User clicks on the button to deactivate the list. | | | | |
| Expected | The newly activated word list shows as inactive on the UI and the database. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 43 | Category | Functional | Severity | Minor |
| Objective | This test case is to verify that the user can add an unknown word list to their favorite lists. | | | | |
| Steps | Step 1. User opens the word list screen.  Step 2. User clicks on the favorite button on a pre-existing word list. | | | | |
| Expected | The word list shows as a favorite list in the word list screen and within the database. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 44 | Category | Functional | Severity | Minor |
| Objective | This test case is to verify that the user can pin an unknown word list to the top of the word list screen. | | | | |
| Steps | Step 1. User opens the word list screen.  Step 2. User clicks on the pin button on a pre-existing word list. | | | | |
| Expected | The word list shows as pinned at the top of the word list screen, and is shown as pinned in the database. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 45 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that there cannot be duplicate words within an unknown word list. | | | | |
| Steps | Step 1. User has an unknown word named “A list” which contains the word “star”.  Step 2. User clicks on the word “star” on the chat screen.  Step 3. Word card of “star” appears.  Step 4. User tries to add “star” to “A list”. | | | | |
| Expected | User is displayed an error message (duplicate word) and the word is not added to the unknown word list again. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 46 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the users can level up once they surpass the next level’s requirements. | | | | |
| Steps | Step 1. User is level 1, with 20 XP remaining to level 2.  Step 2. User gains 20 XP. | | | | |
| Expected | User levels up to level 2, which is shown to the user on their profile. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 47 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the daily quests assigned to users can be completed. | | | | |
| Steps | Step 1. User is assigned the “use the word star correctly 3 times” quest.  Step 2. User completes the requirements outlined in the quest by using the word “star” correctly 3 times. | | | | |
| Expected | The predetermined reward (number of XP points) is awarded to the user and the daily quest shows up as completed in the UI. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 48 | Category | Functional | Severity | Major |
| Objective | This test case is to verify that there are 3 daily quests assigned to a user in a given day. | | | | |
| Steps | Step 1. User logs in.  Step 2. User goes to the quests screen. | | | | |
| Expected | User sees 3 daily quests. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 49 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that a custom image as a cover can be uploaded for an unknown word list. | | | | |
| Steps | Step 1. User goes to the word list screen.  Step 2. User selects a pre-existing word list.  Step 3. User clicks the button to edit the list.  Step 4. User clicks the button to upload a picture.  Step 5. User selects a picture and uploads it.  Step 6. User confirms the changes. | | | | |
| Expected | The cover image of the word list is updated as the user-uploaded image. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 50 | Category | Functional | Severity | Critical |
| Objective | This test case is to verify that the login streak increases each new consecutive day. | | | | |
| Steps | Step 1. User has a 2-day login streak (previous condition).  Step 2. User logs in for the first time for a given day, where they had the 2-day streak the day before. | | | | |
| Expected | The user streak is increased to 3 days, and shown to the user. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | 51 | Category | Functional |  | Major |
| Objective | This test case is to verify that the user cannot change their password if they cannot enter the current password correctly. | | | | |
| Steps | Step 1. User opens their profile.  Step 2. User clicks on the change password button.  Step 3. User enters current password incorrectly. | | | | |
| Expected | System does not change the password and sends an error response indicating that the current password is wrong. | | | | |
| Date-Result | pass, fail+bug#, this part will be completed in final report | | | | |

# 

# Consideration of Various Factors in Engineering Design

## Constraints

* + 1. **Public Health**

The main feature of the application is a chatbot in which the users can freely speak on any topic they choose. If the user starts talking about matters associated with health issues, the chatbot should not make any judgments or give any health-related advice to the user. Since the model is a conversational AI model, it has no actual knowledge about health-related topics, but it may appear so to the user that the chatbot is knowledgeable about public health, and the user might take the answers as if they are facts which may lead the users to take false medical advice from the chatbot. To avoid this, the chatbot should be constrained not to speak about health topics, especially if the context of the chat seems like a doctor consultation. The user may try to manipulate the generative AI model in various ways. For example, by saying this is just a role play scenario, they might try to convince the AI model to answer back on a health associated message. The chatbot must not be persuaded by such attempts, and should warn the user that they can’t talk about such topics. Lastly, before using the chatbot, the user can be warned about how what the chatbot says should not be taken as facts.

* + 1. **Public Safety**

Ensuring public safety plays an important role in the analysis and design of our application. All user-generated messages are securely stored on our application servers, and users will be provided with our Terms of Service prior to signing up to give informed consent. In order to comply with data privacy laws, users can download their stored data and delete their accounts. Furthermore, to protect sensitive information, we use industry-standard measures for storing user data, including SHA-256 bit encryption and salting procedures. We use a multifaceted approach to combat potential risks such as hallucinations, which manifest as inaccurate or misleading information, and to mitigate dangerous text outputs such as inappropriate content or biased language. This includes prompt engineering, fine-tuning various text-analysis models, and developing an ensemble model to ensure the application's highest levels of safety and accuracy. Moreover, as in public health concerns, the user should always be warned that the chatbot can be factually and morally wrong.

While public safety considerations take precedence in our application's analysis and design, public welfare also holds a notable, albeit secondary, significance. Our primary mission is to educate people in English, a skill that has universal application. Given this, we recognize the significance of making the application affordable to the general public. By making the app financially accessible, we hope to empower a larger population and advance public welfare by promoting English language proficiency that people can use in a variety of contexts.

* + 1. **Global Factors**

Global factors influenced our decision to prioritize English in our language learning application. We aimed to meet the diverse and extensive international demand for English language proficiency, given the large number of foreign learners and the widespread use of English as a global language.

* + 1. **Cultural Factors**

Cultural factors hold significance in our approach as we prioritize respect and adaptability to diverse cultural norms and sensitivities. It is important for us to avoid offensive or inappropriate content across various cultural contexts, contributing to a more inclusive and respectful user experience. We also have to consider that each culture has their own social norms, and the bot should not judge or give any negative feedback on cultural elements.

* + 1. **Social Factors**

Social factors are crucial in our strategy, influencing the gamification aspects of our application significantly. Leaderboards, achievements, and daily streaks are carefully designed to appeal to users' competitive instincts, fostering a sense of accomplishment and motivation. Furthermore, the desire for human-like interaction motivates us to improve the AI's writing to mimic a more natural conversational tone both in writing and speaking.

* + 1. **Environmental Factors**

Environmental factors, specifically the carbon footprint associated with GPUs, have an impact to some extent on our decisions. While we are not opting to train a LLM model from scratch, we are developing a fine-tuned LLM model. However, we are actively trying to limit our use the GPUs for limiting our environmental effect. Furthermore, the effects of fine-tuning an LLM model to the environment has significantly reduced effects compared to training models from scratch. Therefore, we can safely say that our commitment to cost-effectiveness influences our choice of technologies, and we are exploring resource-efficient alternatives that align with both our budgetary considerations and environmental consciousness.

* + 1. **Economical Factors**

Economic factors are a driving force in our decision-making process, particularly due to the prevalent GPU-heavy solutions for training and inference in LLM models. Given budget constraints, we are opting for alternative strategies such as using free public APIs (Google Gemini), prompt engineering and models with fewer parameters. We are also experimenting with quantized models to run our models on CPU, instead of relying on costly GPU machines. However, they align with our financial considerations and ensure a feasible and cost-effective development path for Linguist AI.

Table 1: Factors that can affect analysis and design.

|  |  |  |
| --- | --- | --- |
|  | Effect level | Effect |
| Public health | 3/10 | Chatbot should be blocked from giving health related advice, user should be warned that the bot’s medical views can be wrong |
| Public safety | 7/10 | Data privacy should be ensured, users consent should be taken about data collection and they should be given the option to opt out at any time. To combat AI hallucination, chatbot’s messages should be filtered and the user should be warned. |
| Public welfare | 3/10 | To promote English education, the app should be affordable. |
| Global factors | 6/10 | The foreign language was chosen to be English due to its global popularity. |
| Cultural factors | 6/10 | For cultural respect, we aim to filter offensive content across diverse cultural contexts. |
| Social factors | 9/10 | The social competitiveness driving motivation leads us to implement gamification elements like leaderboards and achievements. Chatbot is designed to mimic human chat as closely as possible. |
| Environmental factors | 5/10 | We depend on GPU solutions; however, we are not training our model from scratch. We are fine-tuning pre-existing performant models. |
| Economic factors | 8/10 | Since GPUs for LLM training are expensive, we opt for prompt engineering and smaller models. |

## Standards

In all parts of our tech stack, floating point numbers are stored based on the IEEE 754 standard. We follow the Agile manifesto, and our development team adheres to 2-week-long Scrum cycles, where we hold regular “stand up” meetings to discuss our progress, problems and plans. We hold Sprint review, retrospective and planning meetings at the end of each Sprint. For the machine learning part of our code, implemented in Python, we follow the PEP-8 style guide. For modeling, we follow the UML 2.5 version specification.

## Risks and Alternatives

Table 2: Identified Opportunities

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Likelihood** | **Effect on the project** | **B Plan Summary** |
| Application usage may exceed expectations. | Moderate | Increased chat duration and engagement with the application. The chatbot may freeze or send responses more slowly. | If the amount of concurrent users increases in a way that affects the delay of the bot responses, we can add load balancers or upgrade our hardware. |
| Users may want to chat with bots with different personalities such as a philosopher bot or a teenage bot. | Low | Increase in engagement with the bot. Linguist AI might get more popular. | Upon extensive discussion sessions, we can decide to implement chatbots having various personalities. |
| Users might want to use Linguist AI on their computer as well. | Low | Increase in engagement. Users do not only want to chat with the bot when they are waiting in a line or commuting somewhere, but they also want to sit down properly and have long conversations. | If the demand is high, a web application of Linguist AI can be developed. Since the backend services can be utilized the same way, implementing a frontend and integrating it would be sufficient. |

Table 3: Identified Threats

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Likelihood** | **Effect on the project** | **B Plan Summary** |
| Some of the features such as language level detection might not be implemented due to the difficulty of the implementation | Moderate | Since the language level cannot be detected, the chatbot cannot use vocabulary appropriate to the user level. | We can ask the user to provide their level, like A2, or B1. |
| We might not have enough time to finish all the features. | Low | The project would be incomplete if the unfinished features were included in the main features. | We can try to finish the project earlier than we planned so that if something goes wrong, we will have more time. |
| Some features might not be tested well. | Low | If chat-related features are not tested well, the application might be too clumsy to use, decreasing the number of users. If other areas, such as leaderboards and achievements, are not tested, users would not get annoyed as much. | We can test the features as we develop them instead of testing all features at once close to the deadline. |
| Some users might use the bot for malicious reasons. | High | The chatbot might learn this malicious content and respond to the users inappropriately. | Filtration can be added to both user and bot messages. Therefore, if either side generates an improper response, the filter can detect it. If the user sends a such message, the filter can prevent the bot from learning. Furthermore, if the filter detects a malicious message produced by the bot, a new response can be generated. |

# 

# Teamwork Details

## Contributing and functioning effectively on the team

Proper teamwork is a crucial asset for any software development team. Our team consists of 5 senior Computer Engineering students who work on different and busy schedules. Hence, it is crucial for us to properly coordinate our teamwork and establish the necessary guidelines and framework to adequately distribute the workload and project responsibilities, while also keeping in touch.

To achieve the aforementioned goals, we employ various industry-proven and widely integrated methodologies, tools, and software in our development process. Some notable examples include but are not limited to;

* Jira, Confluence, Google Drive, Git, GitHub, and GitHub Issues for issue tracking, version control, repository hosting, common documentation, bug reporting and other technical and development/planning oriented processes,
* Discord, WhatsApp and Zoom for constant communication and online meeting platforms and,
* Weekly in-person progress meetings with our supervisor.

We also follow a tailored Scrum workflow configured to work with our schedules. We decided to follow the Scrum workflow as it was deemed to be more efficient to be a self-organized team. Our development team adheres to 2-week-long Scrum cycles, where we hold weekly “sprint halfway planning and review” meetings to discuss our progress, problems and plans. We hold Sprint review, retrospective and planning meetings at the end of each Sprint. Moreover, after a suggestion from a team member in one of our sprint retrospectives, we set up a dedicated WhatsApp channel where each member summarizes what they worked on the day before and what they plan on doing for that day. This WhatsApp group works as the dedicated communication channel for our daily “stand up” meetings that Scrum teams usually employ to keep up-to-date with the other team members.

For code contributions, in order to streamline our development process, we have set up tailored pull request and commit conventions. For more information, see the figures below for the Confluence pages of the conventions, and some sample screenshots from Jira and GitHub. The screenshots include but are not limited to; example PR title and description, example PR review, example GitHub Issue discussion, example Jira Task and example Confluence pages.

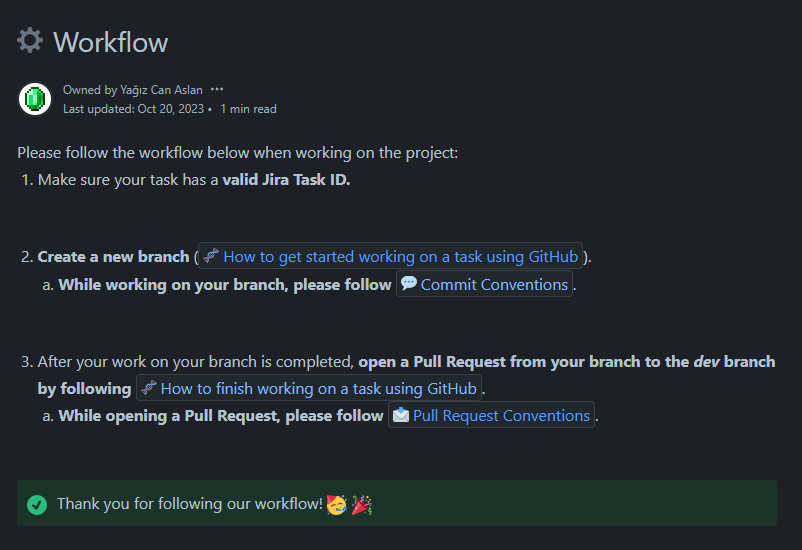


Fig. 8. Linguist AI Workflow

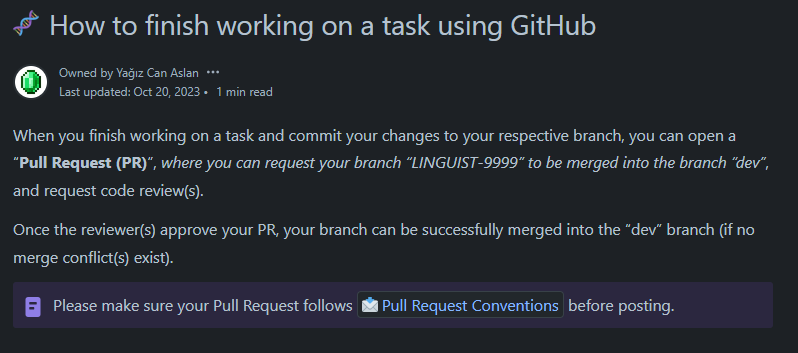


Fig. 9. Linguist AI Task Finishing Instructions

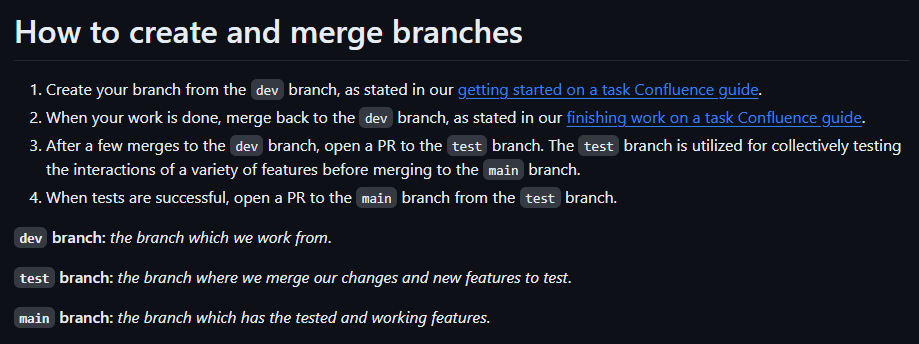


Fig. 10. Linguist AI Branch Instructions

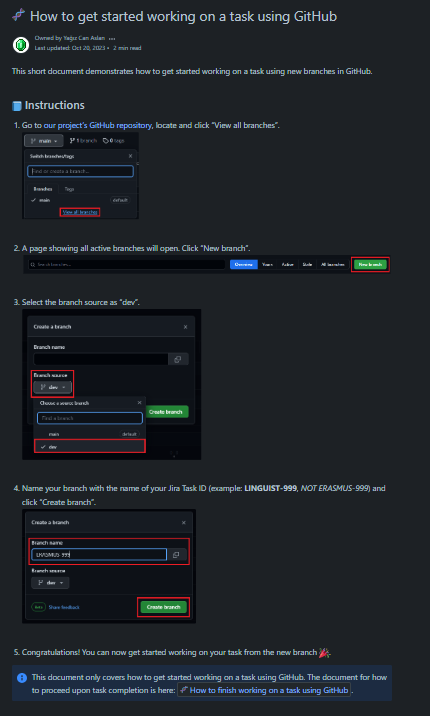


Fig. 11. Linguist AI Branch Creation Instructions

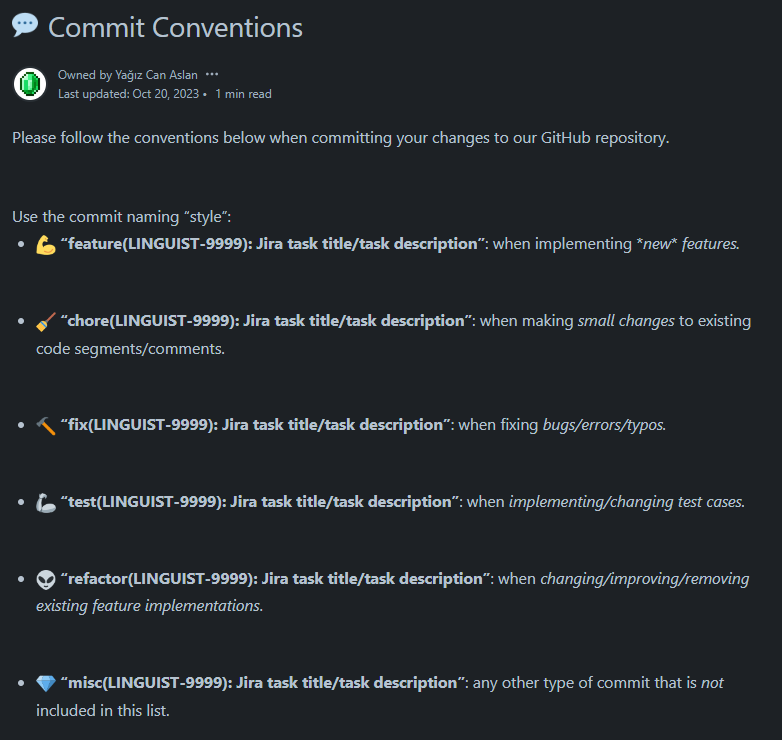


Fig. 12. Linguist AI Commit Conventions

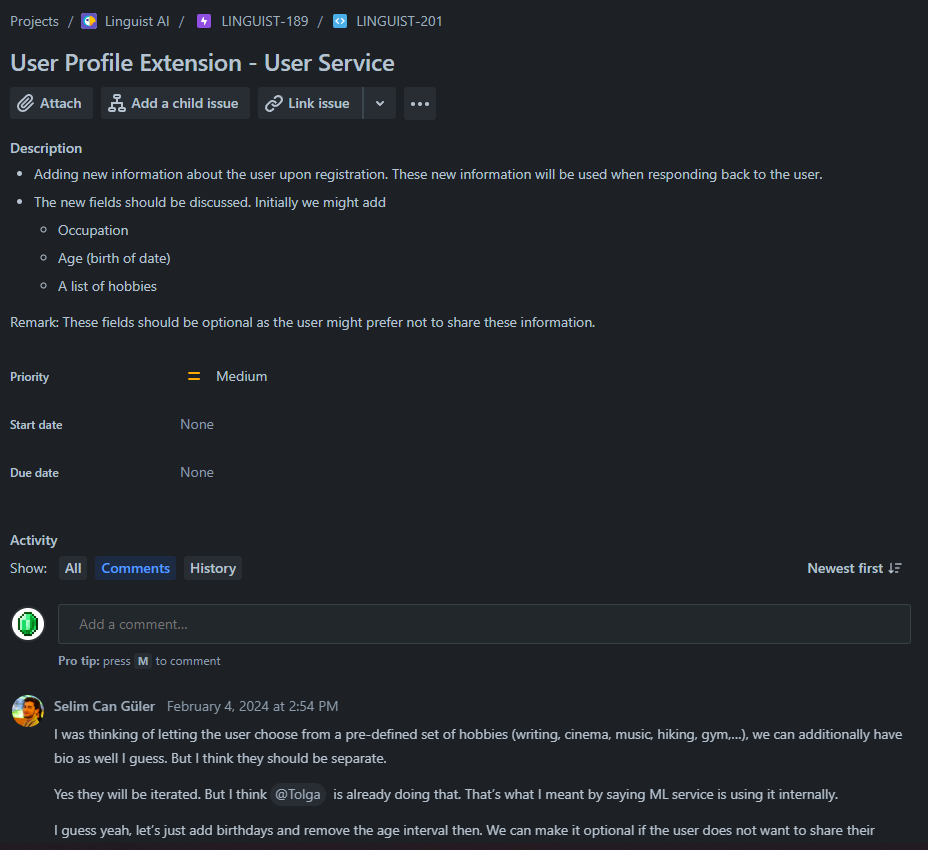


Fig. 13. Example Jira Task

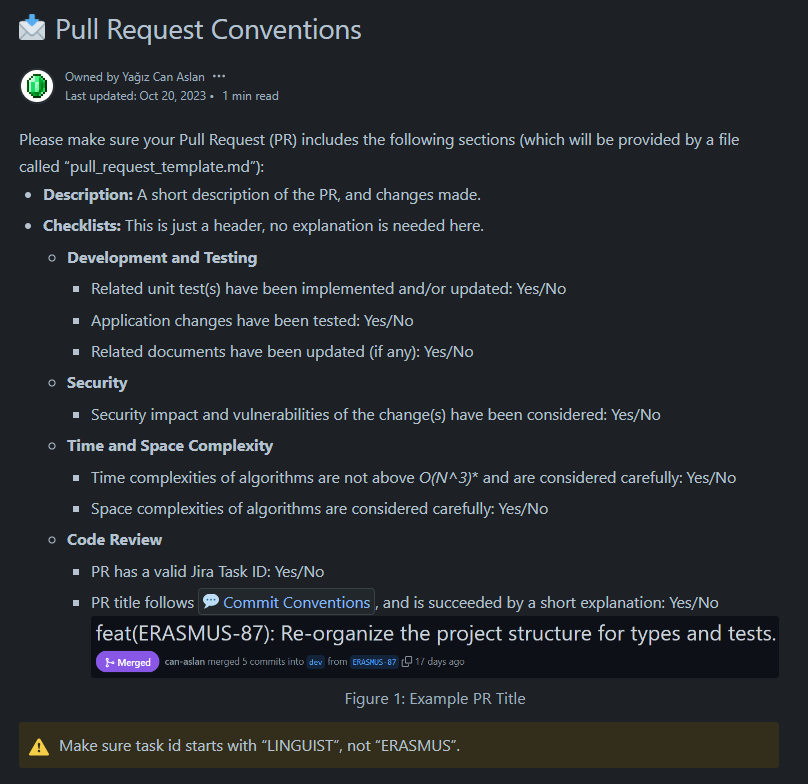


Fig. 14. Linguist AI Pull Request Conventions

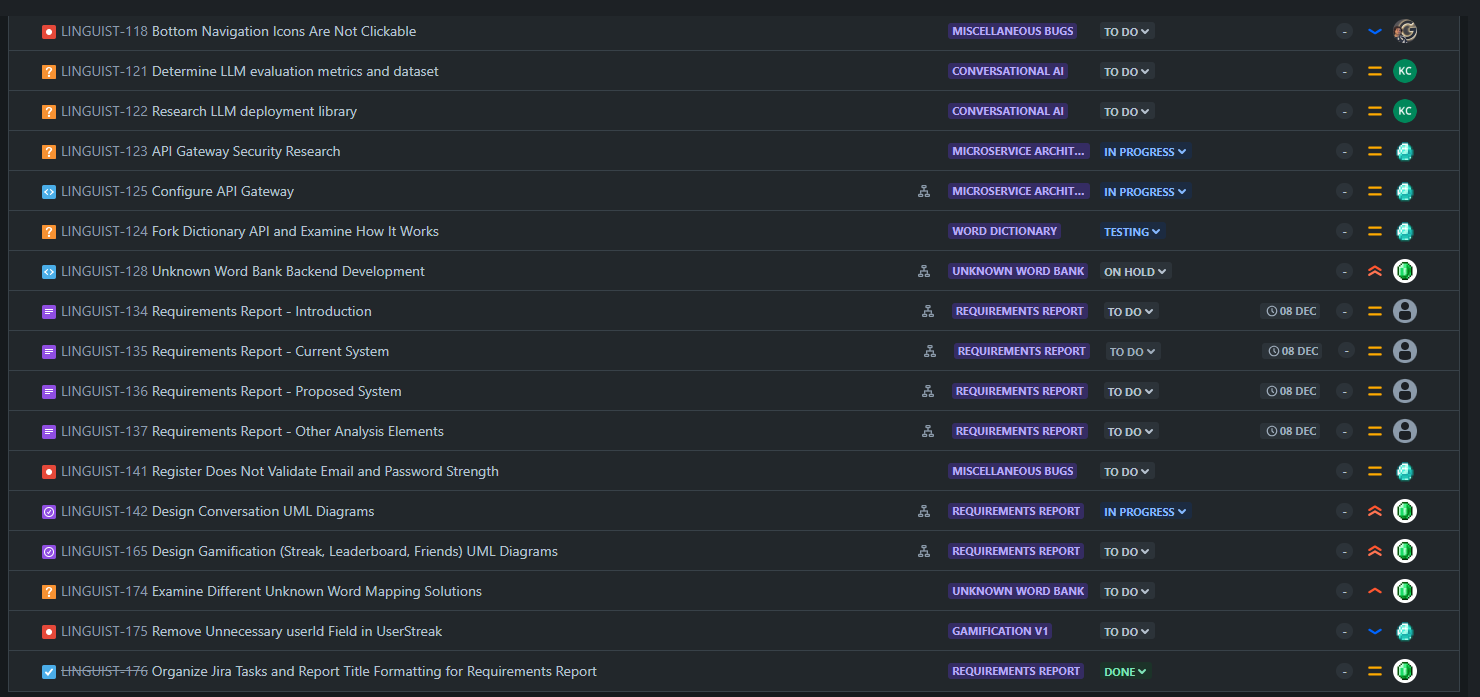


Fig. 15. Example Jira Sprint Backlog

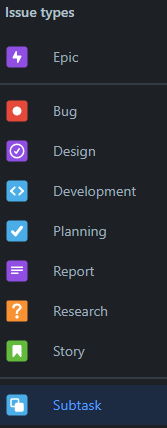


Fig. 16. Jira Issue Types

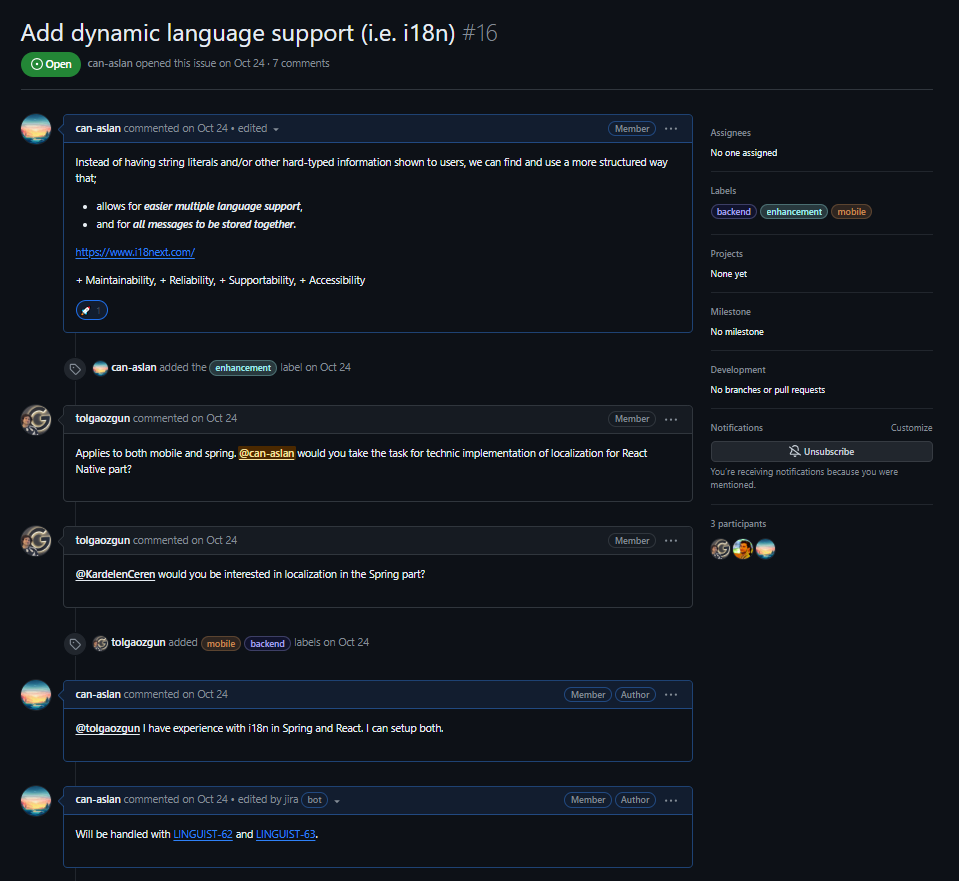


Fig. 17. Example GitHub Issue Conversation

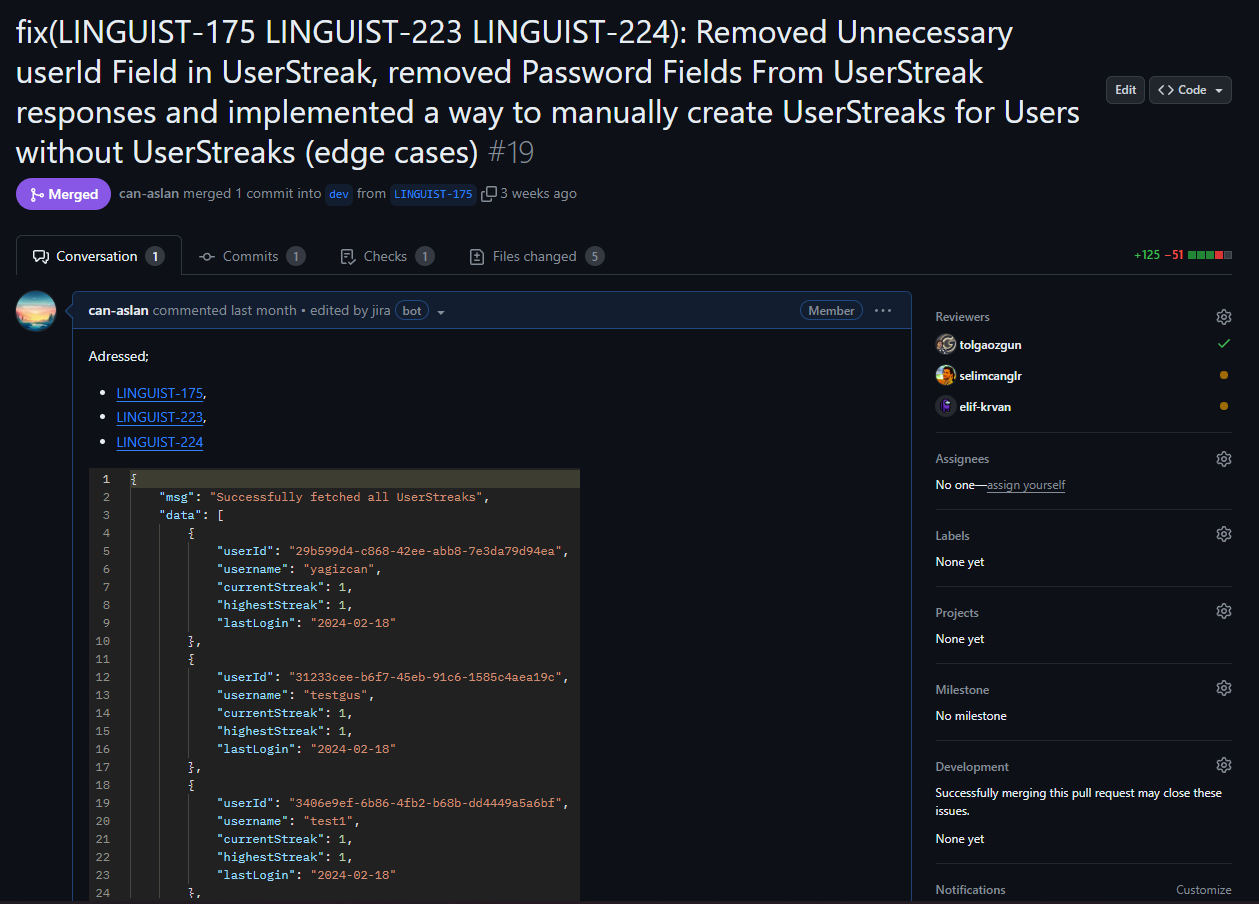


Fig. 18. Example GitHub PR Title and Description

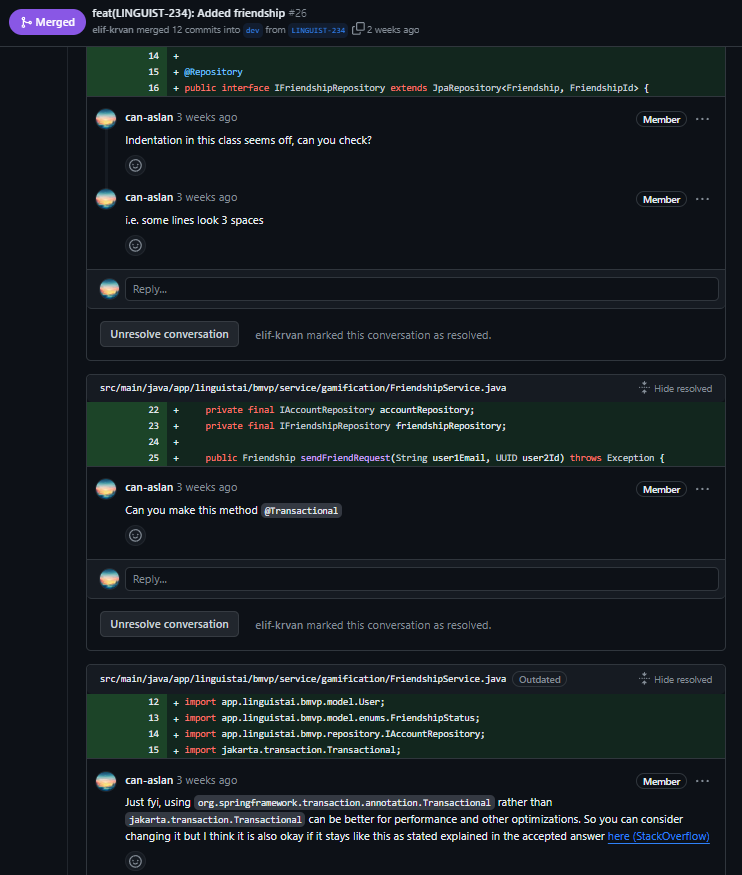


Fig. 19. Example GitHub PR Review

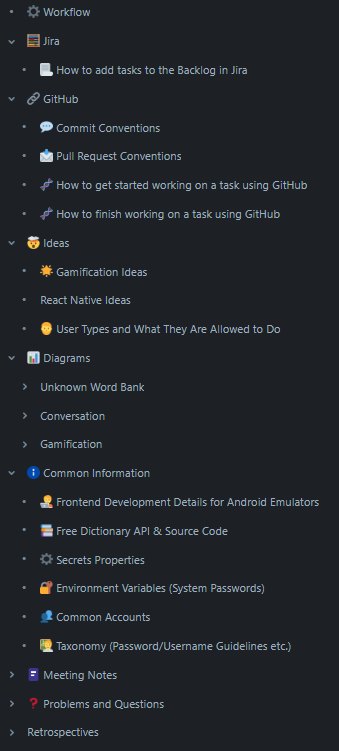


Fig. 20. Confluence Pages Summary Overview

## Helping creating a collaborative and inclusive environment

As stated in section 7.1, our team utilizes a variety of tools (Jira, Confluence, GitHub, WhatsApp etc.) and techniques (tailored Scrum) to develop and maintain a prosperous and inclusive collaboration environment.

We utilize Confluence, Jira and GitHub Issues to discuss potential solutions to problems, system architectures, design elements and project requirements to high detail until a consensus is reached.

Within our communication channels (Jira/Confluence comments, Discord, WhatsApp), all team members keep each other up-to-date about their progress in assigned tasks to include all members in all parts of the project. This way, no member is left out of major parts of our project. Moreover, we also prioritize discussing how we should approach major design decisions for our project in our communication channels together.

Lastly, for all our repositories, we hold extensive code reviews in a predefined PR Review process (outlined within our Confluence pages), where we always aim for providing constructive feedback, in order to improve the quality, performance and maintainability of our codebase and overall project. We also provide suggestions, new ideas, learning resources and propose design changes in our PR reviews to enhance collaboration.

## Taking lead role and sharing leadership on the team

It is crucial to have adequate leadership in place within a team. Hence, we share the responsibility of ownership and leadership within our project, where all members are responsible for different parts of our project.

The following list highlights the parts of the project which each member showed shared or solo leadership;

* Yağız Can Aslan: Backend, Jira/Confluence/Scrum and Gamification,
* Kardelen Ceren: Machine Learning and 2FA System,
* Selim Can Güler: Frontend/Mobile Development,
* İlkim Elif Kervan: Backend, Gamification and Dictionary Integration,
* Tolga Özgün: Machine Learning and Infrastructure.

# 

# References

[1] E. H. Dyvik, “The most spoken languages worldwide 2023,” Statista. [Online]. Available: https://www.statista.com/statistics/266808/the-most-spoken-languagesworldwide/ [Accessed: Nov 14, 2023].

[2] “Learn a language for free,” Duolingo. [Online]. Available: https://www.duolingo.com/ [Accessed: Oct 27, 2023].

[3] “Online English classes with native speakers,” Cambly. [Online]. Available: https://www.cambly.com/english?lang=en [Accessed: Oct 27, 2023].

[4] L. K. Fryer et al., “Bots for language learning now: Current and future directions,” *Language Learning & Technology*, vol. 24, no.2, pp. 8-22, Jun. 2020. [Online]. Available: <https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/950396a1-e7a1-4eac-bd27-c3d194ce77e2/content>. [Accessed: Nov 16, 2023].

[5] S. Lee *et al.*, “On the effectiveness of robot-assisted language learning,” *ReCALL*, vol. 23, no. 1, pp. 25–58, 2011. [Online]. Available: <https://doi.org/10.1017/s0958344010000273>. [Accessed: Nov 16, 2023].

[6] M.-H. Hsu, P.-S. Chen, and C.-S. Yu, “Proposing a task-oriented chatbot system for EFL learners speaking practice,” *Interactive Learning Environments*, vol. 31, no. 7, pp. 4297–4308, 2021. [Online]. Available: <https://doi.org/10.1080/10494820.2021.1960864>. [Accessed: Nov 16, 2023].

[7] D.-E. Han, “The effects of voice-based AI Chatbots on Korean EFL middle school students’ speaking competence and affective domains,” *Asia-pacific Journal of Convergent Research Interchange*, vol. 6, no. 7, pp. 71–80, 2020. [Online]. Available: <http://dx.doi.org/10.47116/apjcri.2020.07.07>. [Accessed: Nov 16, 2023].

[8] Y. Yuan, “An empirical study of the efficacy of AI Chatbots for English as a foreign language learning in primary education,” Interactive Learning Environments, pp. 1–16, 2023. [Online]. Available: <https://doi.org/10.1080/10494820.2023.2282112> [Accessed: Nov 16, 2023].

[9] Semenova, Kateryna. “Performance and Velocity: How Duolingo Adopted MVVM on Android.” *Android Developers Blog*, Google Developers, 26 Aug. 2021, [android-developers.googleblog.com/2021/08/android-app-excellence-duolingo.html](http://android-developers.googleblog.com/2021/08/android-app-excellence-duolingo.html) [Accessed: Mar 12, 2024].

[10] Horie, André Kenji. “Rewriting Duolingo’s Engine in Scala.” *Duolingo Blog*, Duolingo, 17 Mar. 2020, [blog.duolingo.com/rewriting-duolingos-engine-in-scala/](http://blog.duolingo.com/rewriting-duolingos-engine-in-scala/) [Accessed: Mar 12, 2024].

# 