



Bilkent University
Department of Computer Engineering

Senior Design Project
T2314
Linguist

Final Report

22001943, Yağız Can Aslan, can.aslan@ug.bilkent.edu.tr
22003017, Kardelen Ceren, kardelen.ceren@ug.bilkent.edu.tr
22002811, Selim Can Güler, selim.guler@ug.bilkent.edu.tr
22002223, İlkim Elif Kervan, elif.kervan@ug.bilkent.edu.tr
22003850, Tolga Özgün, tolga.ozgun@ug.bilkent.edu.tr

Supervisor: Halil Altay Güvenir
Course Instructors
Atakan Erdem
Mert Bıçakçı

13.05.2024

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

Abstract.....	3
1 Introduction.....	4
1.1 Purpose of the system.....	5
1.2 Definitions, acronyms, and abbreviations.....	5
1.3 Overview.....	6
2 Requirements Details.....	7
2.1 Functional Requirements.....	7
2.1.1 User.....	7
2.1.2 Chatbot.....	8
2.2 Non-Functional Requirements.....	8
2.2.1 Usability.....	8
2.2.2 Reliability.....	9
2.2.3 Performance.....	10
2.2.4 Supportability.....	11
2.2.5 Scalability.....	11
2.2.6 Security & Privacy.....	11
2.3 Pseudo Requirements.....	12
3 Final Architecture and Design Details.....	13
3.1 Overview.....	13
3.2 Subsystem decomposition.....	14
3.3 Hardware/software mapping.....	14
3.3.1 Mobile Application.....	15
3.3.1.1 Microphone and Speaker Access.....	15
3.3.1.2 Touchscreen Support.....	15
3.3.1.3 Notifications Support.....	15
3.3.1.4 Image Gallery - Filesystem Access.....	15
3.3.2 Machine Learning Subservice.....	15
3.3.2.1 GPUs for Fine-Tuned LLM:.....	15
3.3.2.2 CPUs for Quantized LLM:.....	15
3.4 Persistent data management.....	16
3.5 Access control and security.....	16
3.5.1 Authentication and session management.....	16
3.5.2 Layered authentication checks.....	17
3.6 Subsystem services.....	18
3.6.1 Client Subsystem.....	18
3.6.2 Backend Subsystem.....	19
4 Development/Implementation Details.....	23
System Design Process.....	23
Architecture Implementation & Development Process.....	23
Brainstorming During Development.....	24
Brainstorming After Development.....	24
5 Test Cases.....	25

6 Maintenance Plan and Details.....	44
7 Other Project Elements.....	44
7.1 Consideration of Various Factors in Engineering Design.....	44
7.1.1 Constraints.....	44
7.1.1.1 Public Health.....	45
7.1.1.2 Public Safety.....	45
7.1.1.3 Global Factors.....	45
7.1.1.4 Cultural Factors.....	45
7.1.1.5 Social Factors.....	46
7.1.1.6 Environmental Factors.....	46
7.1.1.7 Economical Factors.....	46
7.1.2 Standards.....	47
7.2 Ethics and Professional Responsibilities.....	47
7.2.1 Risks and Alternatives.....	48
7.3 Teamwork Details.....	50
7.3.1 Contributing and functioning effectively on the team.....	50
7.3.2 Helping creating a collaborative and inclusive environment.....	60
7.3.3 Taking lead role and sharing leadership on the team.....	60
7.3.4 Meeting objectives.....	61
7.4 New Knowledge Acquired and Applied.....	72
8 Conclusion and Future Work.....	74
8.1 Conclusion.....	74
8.2 Future Work.....	74
8.3 Potential Business Strategies.....	75
8.4 References.....	76

Final Report

T2314: Linguist

Abstract

Linguist is a revolutionary language learning mobile application, powered by AI. Traditional learning methods rely on predefined material, often ineffective and disengaging. Linguist overcomes this by shifting the learning to learner-specific conversations with different AI personas. Linguist's AI learns relevant information about the learner over time (hobbies, profession, life events, likes, dislikes etc.), while closely monitoring each conversation to understand the gaps in the learner's vocabulary. This information is used in each message, regularly subjecting learners to unknown vocabulary to enhance understanding. Gamification elements create friendly competition, which enhances the effectiveness of the learning process. This builds a seamless learning experience and enables learners to practice the topics they choose, at any time.

1 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.

Linguist 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 includes gamification elements such as achievements, leaderboards, and daily streaks. The main focus of Linguist 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 enables users to learn English tailored to their level by eliminating artificial learning items such as flashcards and textbooks and offering a sincere conversational environment.

1.1 Purpose of the system

One of the primary goals of Linguist 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, Linguist 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 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.

Linguist 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, Linguist makes language practice available to a diverse range of learners.

1.2 Definitions, acronyms, and abbreviations

XP: Experience Point,

JSON: JavaScript Object Notation,

JWT: JSON Web Token,

AWS: Amazon Web Services,

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 (Turkey's GDPR),

CPU: Central Processing Unit,

GPU: Graphical Processing Unit

1.3 Overview

For those learning English as a foreign language, Linguist 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, Linguist 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.

2 Requirements Details

This section details the functional, non-functional and the pseudo requirements of our project.

2.1 Functional Requirements

This section outlines the specific actions and functionalities that the system performs to meet the needs of its users, detailed through what the user can do and what the chatbot can do.

2.1.1 User

- The user can register by using their email and accepting Linguist's privacy policy.
- The user can see the privacy policy from their profile.
- The user can login using their email.
- The user can change their password after logging in.
- The user is sent an email with a passcode to reset their password if they forget it.
- The user's basic information such as their name, birth date, hobbies and english level is collected during an optional onboarding process.
- The user can have a conversation with the chatbot by texting with the chatbot.
- The user is presented with onboarding messages to orient the user the first time they enter the chat screen.
- The user can also choose to have a conversation with the chatbot by speaking.
- The user can listen to the pronunciation of the chatbot's texts.
- During their conversation with the chatbot, the user can select a word they do not know and add it to their unknown word storage.
- The user can see the meaning of the words they select, along with an example sentence including the word.
- The user can see the list of unknown words they saved so far, with their meanings and example sentences available on click.
- The user can have multiple unknown word lists, and activate the ones they would like the bots and the quizzes to select words from.
- The user can add words they do not know to their lists.
- The user is presented with regular quizzes to determine how well they know unknown vocabulary.
- Based on the user's answers to quizzes, word's confidence level (i.e., how well the user knows that word) is updated.
- The user can see the words the bot is trying to teach on the chat screen.
- The user can access their profile to see and edit their personal information.
- The user has the option to view the likes/dislikes collected by the bot during their conversation and delete them if desired.
- The user can update their profile picture.
- The user can search for other users and see their profiles with their basic information.
- The user can send friend requests to other users and cancel requests.
- The user can accept or reject friend requests.

- The user is assigned daily quests such as using a specific word a certain number of times in a conversation or sending a specified number of messages.
- The user earns experience points when they finish quests or log in for the first time each day.
- The user's level increases as they accumulate experience points.
- The user can view the leaderboard to compare themselves globally or with their friends in terms the experience points they gained.
- The user can see their daily log-in streak on a pop-up when they enter their profile.
- The user can see their total word learning progress as a bar graph.
- The user can see the dates they logged in in the last three months in a graph.
- The user can access the store to purchase items and see their purchased items.

2.1.2 Chatbot

- There are multiple chatbots, each with a different personality and difficulty level.
- The chatbot selects a set amount of words from the activated unknown word lists and uses them more frequently.
- The chatbot is given a summary of chat history and adapts to it for further responses.
- The chatbot is given a user profile from the information that the user provided, such as their name, age, hobbies, English level.
- The chatbot generates a user profile based on the exchanged messages. This profile contains user's likes, dislikes, and general information about them such as their occupation.
- The chatbot uses the given and the generated user profile to tailor the messages to the user.
- The chatbot generates multiple choice question quizzes dynamically based on the chosen (active) words.
- When the chatbot receives a sentence which includes a word that is unknown to the user, the system evaluates how well the user employed that word to update the confidence of the user knowing the word.

2.2 Non-Functional Requirements

This section mentions the non-functional requirements under Usability, Reliability, Performance, Supportability, Scalability and Security.

2.2.1 Usability

Linguist is a mobile application. Therefore, the users are able to use our application anywhere and anytime, 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 has an intuitive and simple user interface, a responsive design, is compatible both with Android and IOs and maintains a consistent design. The application also has 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 are able to reach all parts of the application with at most 5 clicks. For ease of use, the navigation bar is visible on all screens except the chat screen based on the user feedback we received, and the back buttons is always be in the same place (top left). As we mentioned in previous reports, we have defined and adhered to the following constraints for usability:

- Each main subsystem, that is chat, word bank, quests, store, leaderboard, is accessible within 2 clicks for every logged-in user.
- Logging in takes 4 steps at maximum with valid credentials. In our app, this corresponds to opening the app, tapping “log in” from the landing page, entering credentials and tapping “log in” from the log in page.
- All back buttons are placed on the top left of the screen, with 16 dp (density-independent pixels) for left and top margins.
- Each main subsystem has a visible bottom navigation.
- All network requests are asynchronous so that users can interact with the front end while the backend response is being generated.
- All in-progress network requests are clearly indicated to the user with a loading visual placed as the closest element to the target component.

2.2.2 Reliability

In order for our application to achieve its goals, the application provides a consistent experience. All systems of our application (backend servers, databases, LLM integration, mobile application etc.) cooperates smoothly without major errors. To provide a smooth experience to our users, all of our systems have at least 90% uptime:

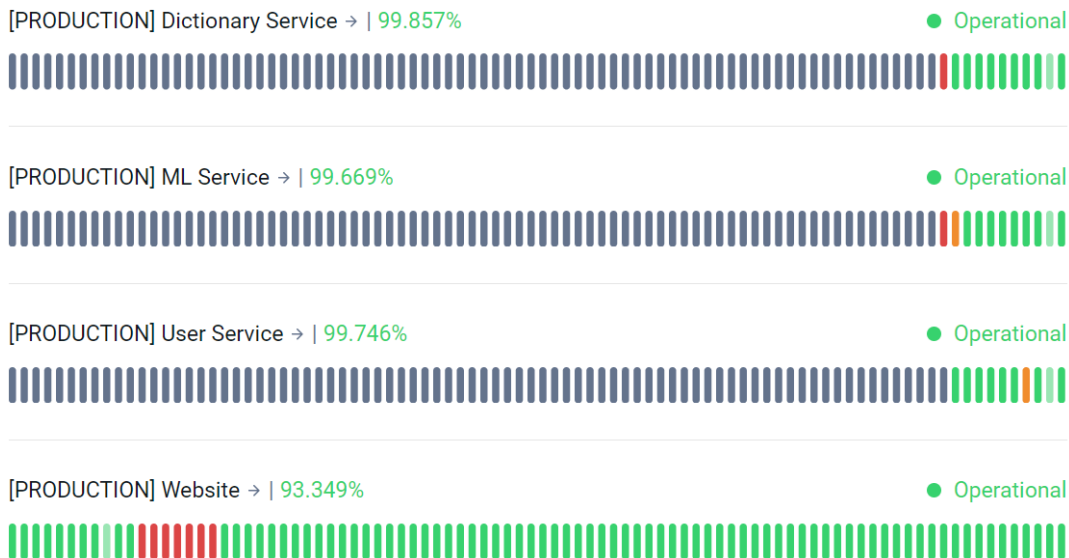


Figure 1. Systems’ uptime

We also utilize Nginx Amplify to monitor VPS Server health and various metrics such as response time, HTTP errors, requests per second and resource utilization to negate any potential issues with the server.

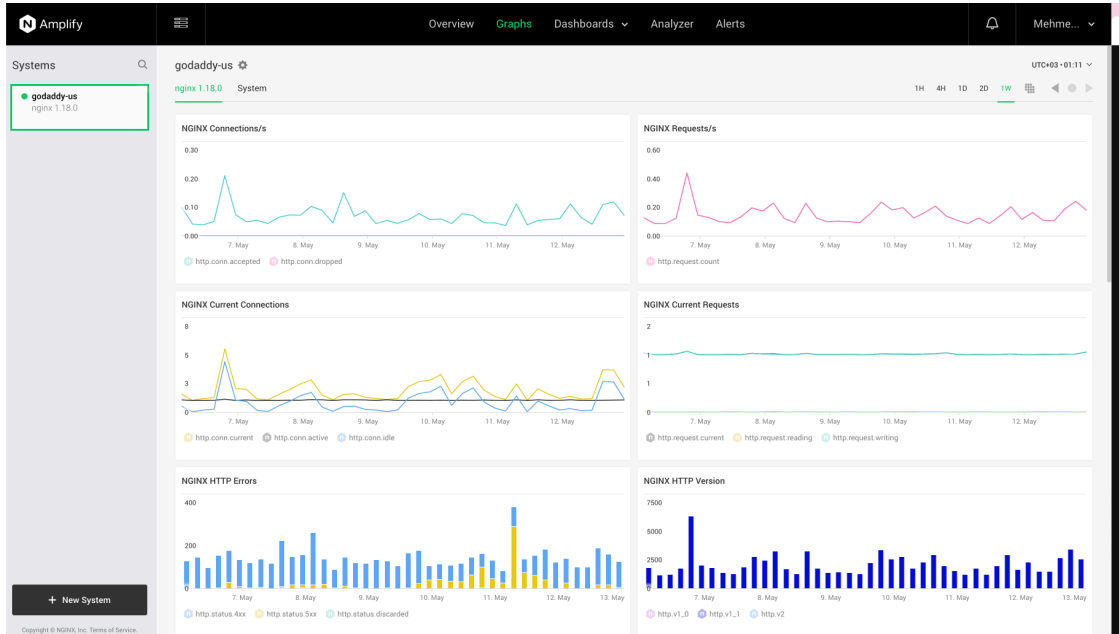


Figure 2. Nginx Amplify Dashboard

The user can continuously use the application without encountering a major error in any step. Moreover, the responses of our conversational AI models are not strange or unsettling, so that the users can know that whenever they use our application, they will be subject to the same experience. We achieve this through the AI's sensitivity settings set to avoid any discriminatory or sexual language, as well as hate speech.

2.2.3 Performance

As stated in section 2.2.1, 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 is strictly under 10 seconds.
- Mean response time of the authentication service is less than 5 seconds.
- Average time to load a page on the mobile application is less than 3 seconds.
- Adding new words to the word list takes less than 2 seconds.
- Loading user profiles takes less than 4 seconds.
- Mean response time of the main subsystems besides chat and authentication is less than 4 seconds.

With these metrics in mind, we've designed a UI that is both performant and responsive. As a result, the application doesn't feel clunky, slow, or outdated.

Additionally, the connections between the subsystems of our project—including backend servers, databases, LLM integration, and the mobile application—occur relatively quickly, facilitating the smooth and responsive experiences mentioned earlier.

2.2.4 Supportability

In order to reach a broader user base, our mobile application is available for Android and iOS. As such, we maintain the application for different operating systems and mobile devices. Our application is designed such that it can 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 supported Android version is 5.0, covering over 98% of the active Android user base, which surpasses Google's recommended minimum of 24.0, ensuring service accessibility to even more users.
- The minimum supported iOS version is 13.0, surpassing Apple's minimum recommendation of 15.0 with 94% of users.

2.2.5 Scalability

As our application is mobile-based, it offers ease of access for new users, facilitating swift downloads and adoption. Consequently, our system infrastructure, including backend servers, databases, LLM integration, and the mobile application itself, is designed to be robust and scalable to accommodate potential surges in user activity and server loads, given enough resources.

Furthermore, our development approach prioritizes flexibility and adaptability, allowing for smooth migration, incorporation of new features, and expansion to distribution channels such as the App Store and Google Play Store, which we are currently using for closed beta testing.

2.2.6 Security & Privacy

Linguist stores user data to provide a specialized learning experience. The stored data may include names, hobbies, interests, passwords, emails, conversation history and many more types of information. Therefore, 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 utilize complex encryption algorithms. Moreover, we follow GDPR (General Data Protection Regulation) and KVKK (Kişisel Verilerin Korunması Kanunu) guidelines closely and seek user consent whenever necessary, to strike a balance between personalization and privacy. We implemented the following procedures for our security purposes:

- Users must accept our privacy policy detailing which data we are collecting and for what purposes before registering. A copy of the privacy policy can be accessed here: <https://Linguist.app/privacy>.
- All passwords must be hashed with bcrypt hashing algorithm.

- All sessions are given a session token by JWT and all requests have a valid access token.
- All users are provided with a refresh token to regenerate the access token when expired.
- Access token is valid for 24 hours.
- Refresh token is valid for 7 days.

2.3 Pseudo Requirements

- Github, Jira and Confluence are being used extensively to maintain the development process of our project.
- React Native is used as a framework to develop our application for the most common mobile operating systems, such as iOS and Android, with a single codebase.
- React Native Platform is used to create responsive layouts for different device sizes and to create layouts that are adaptive to landscape and portrait modes.
- React Navigation is used to create Stack-based routing and bottom tab routing and navigation for the entire application.
- Axios is used to send requests over the Internet and intercept them to prevent unwanted behavior when needed.
- Expo ImagePicker is used when users want to add a profile picture to their profiles.
- Microservice architecture is adopted for server-side development.
- Three main microservices are established: user service, machine learning (ML) service, and dictionary service.
- AWS services, including Polly, Transcribe, S3, DynamoDB, Lambda, and SNS, are integrated into the backend system via AWS API Gateway.
- User service backend is developed using Spring Boot with Java.
- Slf4j is used for logging server activity.
- Relational MySQL database is the main point of storing data.
- ML related data is stored in a PostgreSQL database.
- Amazon AWS S3 is used as a secondary database for non-relational data such as files.
- Java Persistence API (JPA) and Hibernate ORM is used to ease database CRUD operations.
- The backend is deployed to a Virtual Desktop Server serving Ubuntu hosted at GoDaddy.
- Pre-trained Large Language Models (LLMs) such as GPT-4, and Gemini is used for chatting, and profile and quiz generation.
- Large Language Models (LLMs) are deployed on a remote Ubuntu server, running with respective quantization methods to support CPU processes.
- LLM services are served leveraging microservice architecture, and a Python Django application is connected to our API Gateway.
- The API Gateway is also hosted in the same Ubuntu machine, handling the redirection of API calls and authentication.

3 Final Architecture and Design Details

This section highlights our final architecture and design goals. We will give detailed analysis on our subsystems, architecture, methodologies and tools.

3.1 Overview

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

end users are required to use their mobile devices since our software is designed and developed for mobile. Our servers are deployed on a virtual server hosted in GoDaddy. Also, we are utilizing some of the AWS services.

The client-side architecture consists of developments made on the mobile application that will utilize the functionalities offered by the server side. The client-side communicates with our backend system via a custom-built API gateway. Server-side architecture is built based on microservice architecture and currently has three microservices: user service, machine learning (ML) service, and dictionary service. Each of the microservices works independently of each other except for communication necessary for some tasks. Apart from these, we also have some services in AWS that are connected to our backend via AWS API Gateway. We are utilizing Polly, Transcribe, S3, DynamoDB, Lambda, and SNS from Amazon. When our gateway receives a request related to our AWS services, it redirects the requests to AWS API Gateway, otherwise it directs the requests to the related microservice.

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, quests for a user, user experience points, etc. For that purpose, we use PostgreSQL and databases deployed in our servers. Furthermore, S3 is utilized for storing binary data such as images, audio and text files and DynamoDB is used for storing data related to push notifications triggered by SNS.

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. Furthermore, the AWS API Gateway only responds to requests that contain a valid API key. Therefore, it is not possible to use our AWS services without having a secure key.

3.2 Subsystem decomposition

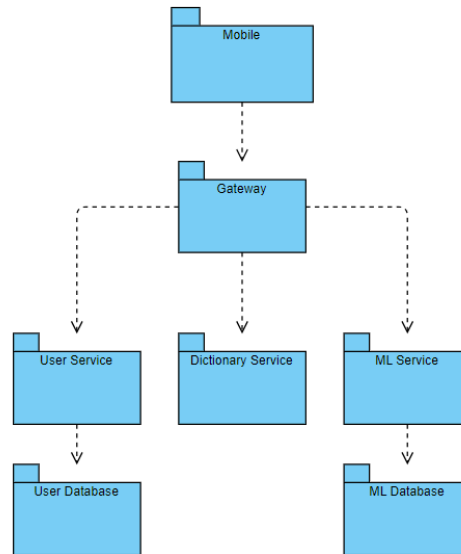


Figure 3. Subsystems of backend hosted in GoDaddy

Linguist 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. 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 such as experience points, leaderboards, friends, and quests 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, word scoring, and multiple question generation. ML service stores its data in a separate PostgreSQL database.

3.3 Hardware/software mapping

Linguist 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.

3.3.1 Mobile Application

Linguist 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.

3.3.1.1 Microphone and Speaker Access

Central to the Linguist 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.

3.3.1.2 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.

3.3.1.3 Notifications Support

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

3.3.1.4 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.

3.3.2 Machine Learning Subservice

3.3.2.1 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.

3.3.2.2 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.

3.4 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.

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, and statistics about users's learning process. In order to efficiently maintain database operations without considering the low-level connection details and native SQL queries, our system uses JPA (Java Persistence API). Furthermore, JPA allows us to easily switch to another database system in the future as it provides interfaces to easily handle queries without writing SQL codes.

Moreover, profile photos, transcription results, and audios generated by AWS Polly are saved to S3. We are sending Base64 encoded data from mobile to the AWS API Gateway which are then converted to byte arrays in Lambda functions before saved to S3 buckets. Also, device and user ids are saved in DynamoDB to be used in push notifications.

3.5 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.

3.5.1 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.

API Keys for Gateway: AWS API Gateway is secured by API keys. The keys are generated in AWS console. When client side sends a request that needs to be redirected to AWS, our Spring API Gateway inserts x-api-key parameter that contains the generated key to the request header. AWS Gateway only accepts requests with valid keys securing our AWS services as all of them are accessible only from the AWS API Gateway. Initially, we planned to send AWS request directly from the client side without letting the Spring gateway handle the requests. However, we immediately changed our minds since this way we would expose our API keys.

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.

3.5.2 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.

3.6 Subsystem services

3.6.1 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 4.

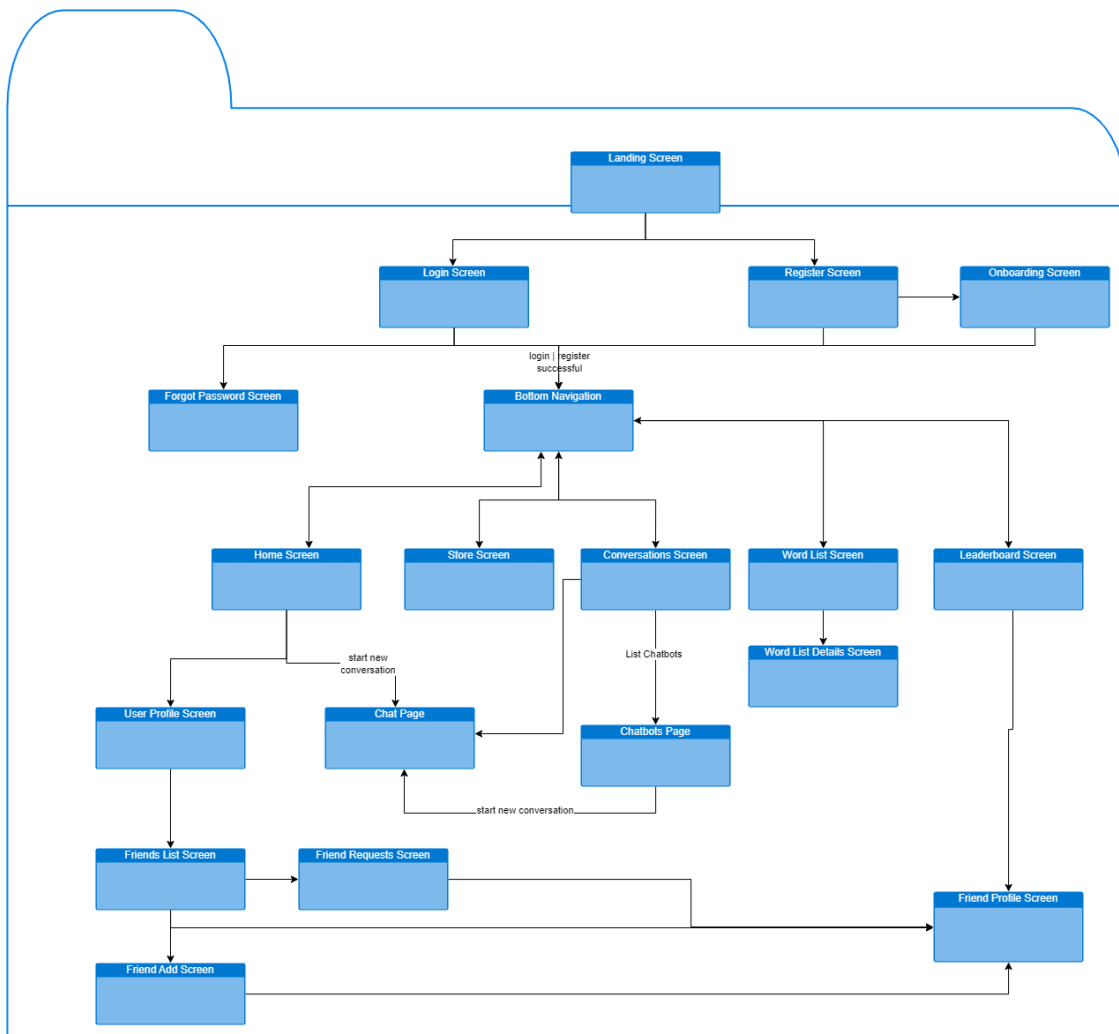


Figure 4. Client architecture diagram

3.6.2 Backend Subsystem

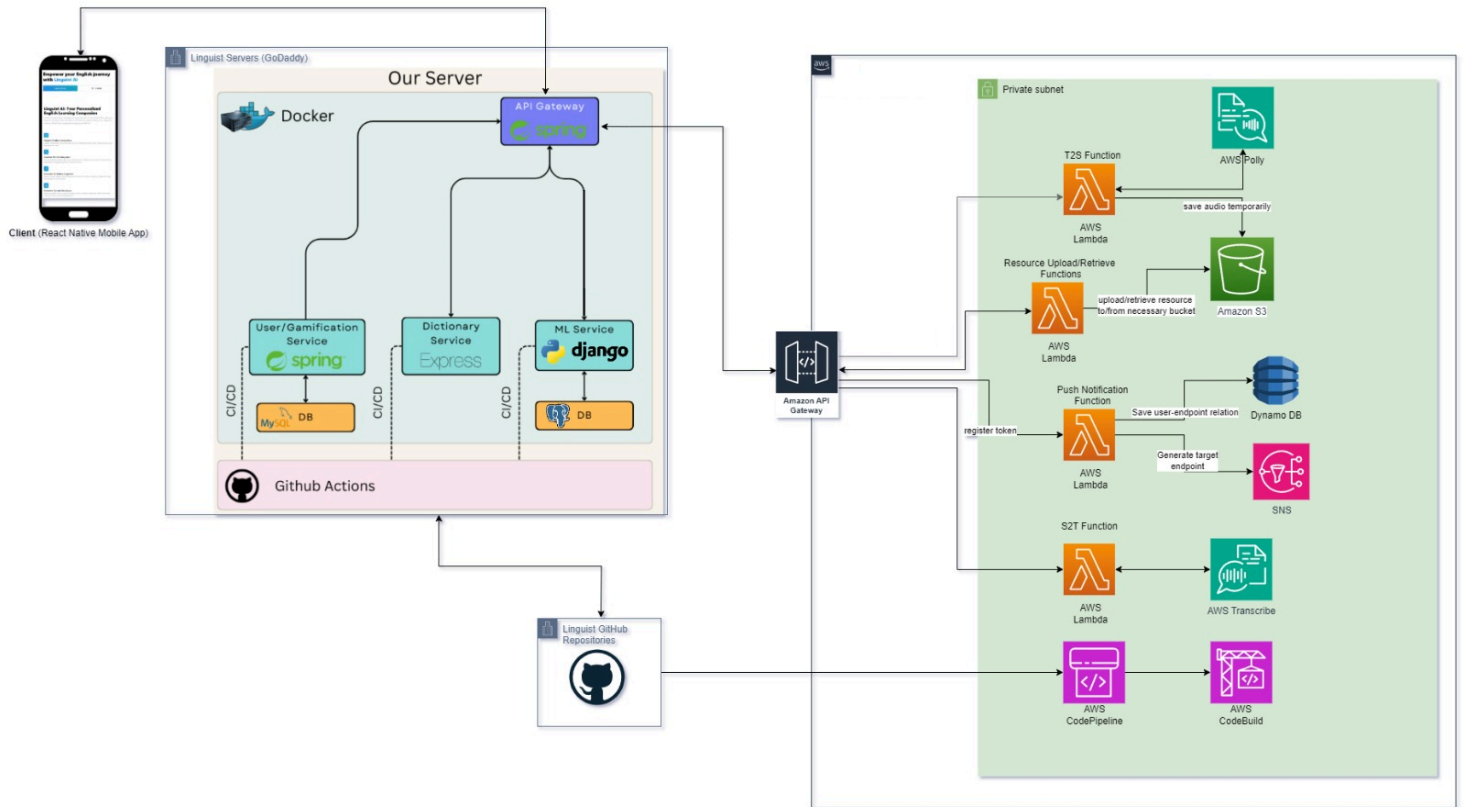


Figure 5. Architecture of the whole system

You can see the high quality of the architecture diagram here: <https://drive.google.com/file/d/1AtK21h0ltZLflhGOM7NTaWm7MPEQjmTt/view?usp=sharing>

The general structure of the backend is given above. All requests are arrived at the gateway and then redirected to the designated microservices or to the AWS API Gateway. Afterwards, related methods are called to execute the logic and database queries are executed if necessary to complete the process, and the response is returned to the client side.

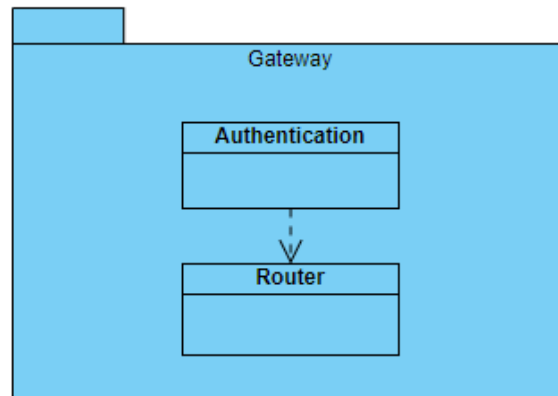


Figure 6. Spring Gateway architecture diagram

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

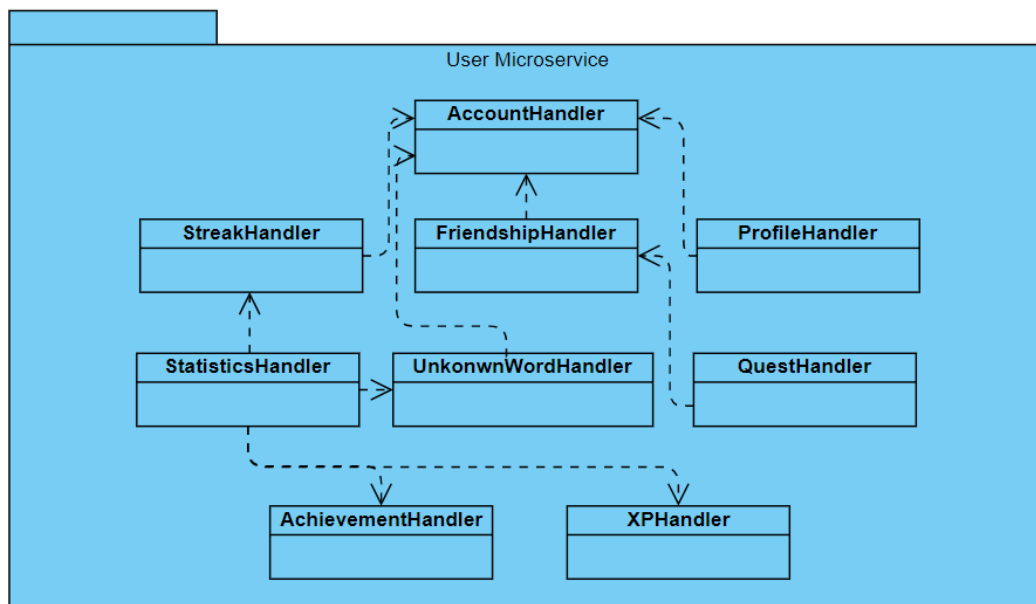


Figure 7. User microservice architecture diagram

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.

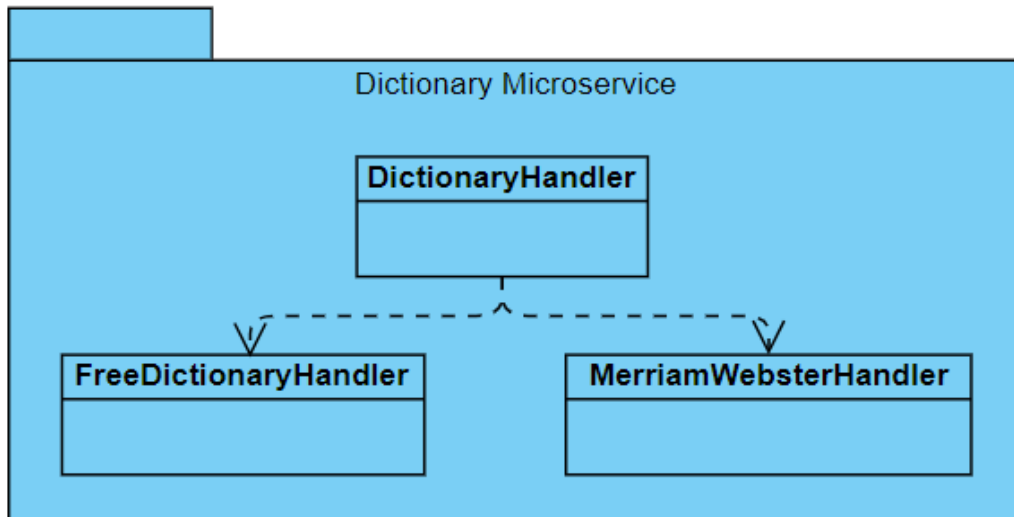


Figure 8. Dictionary microservice architecture diagram

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.

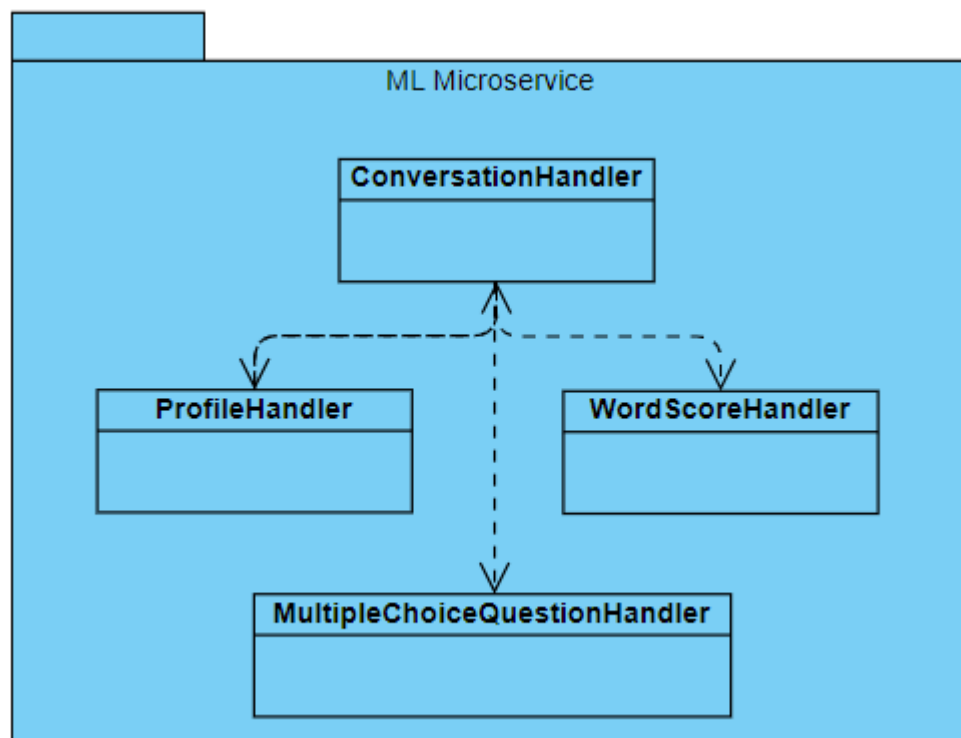


Figure 9. ML microservice architecture diagram

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.

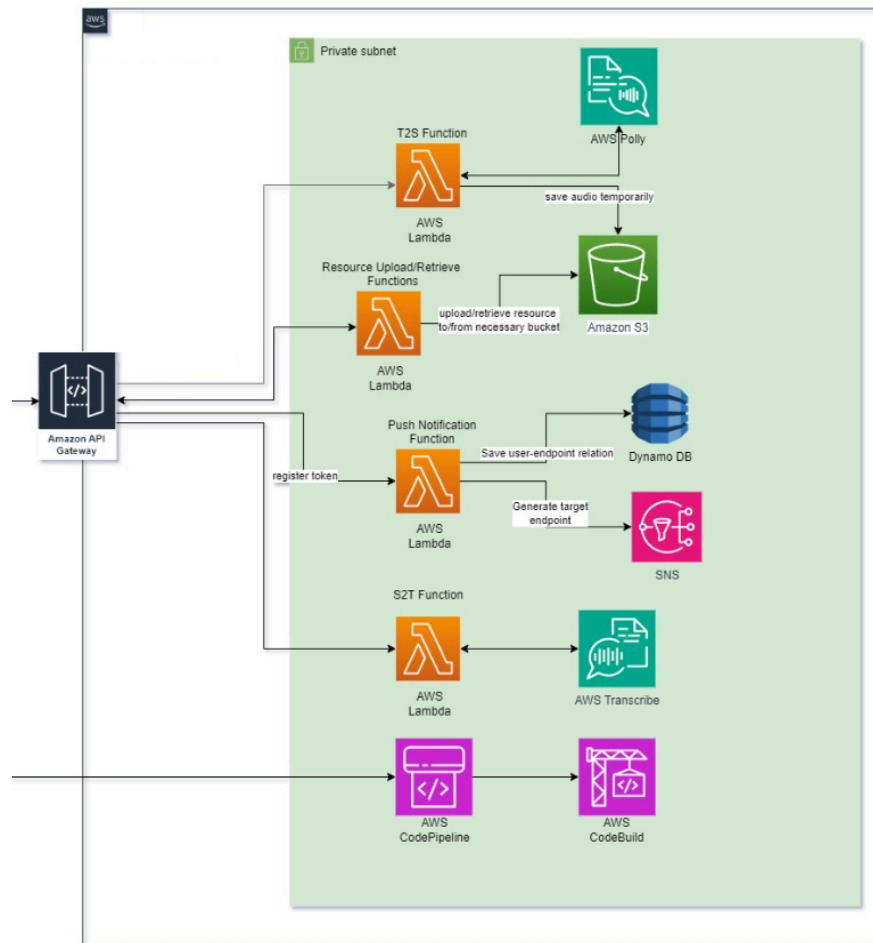


Figure 10. AWS architecture diagram

We are currently using API Gateway, Polly, Transcribe, S3, DynamoDB, SNS, and Lambda functions that run Node from Amazon Web Services and planing to integrate CodePipeline and CodeBuild to our systems until the CS Fair.

When a user clicks on the speaker icon in the chat screen, client sends a request to a Lambda function that sythesizes the audio of the text using Polly. Then, asynchronously the data is saved to an S3 bucket and audio is returned to the client for users to listen.

Transcribe is used for speech to text purposes. If a user clicks on the microphone icon in the chat screen, the audio is saved and converted to Base64 format in mobile. Then, it is sent to another lambda function which saves the audio to S3 and starts transcription job. Then, client sends another request to AWS to retrieve the completed transcript and displays this to the user.

S3 is also used for any kind of file uploading purposes, including profile pictures and word list images. Image upload requests are directed to another lambda that saves images to related S3 buckets.

We are utilizing SNS to send push notifications to Linguist users in order to encourage them to use the app. In DynamoDB ids of the users and their devices are stored and according to these SNS sends notifications to the devices.

4 Development/Implementation Details

Our development process was based on sprints from 1 to 4 weeks depending on our availability during the term. We mostly tried to adhere to SCRUM guidelines during our sprint lifecycle. Before the start of each new sprint we planned our sprint by determining the sprint goal and adding tasks to our backlog. We then assigned those tasks to the relevant group members and started our sprint. Even though we didn't have daily SCRUM meetings, we were constantly in touch through Jira, GitHub, and WhatsApp. At the end of each sprint, we had sprint review and sprint retrospective meetings to get ready for the new sprint.

System Design Process

We have analyzed widely used chatting applications such as WhatsApp, Discord, Facebook and Instagram to determine the best case for our services in terms of scalability, development velocity and stability.

For example, during our database selection process we have first decided to move on with Cassandra, influenced by many articles and especially the implementation of the Discord application. However, during our conversations, we have decided that our team lacked the necessary knowledge for Cassandra development, and we have decided the use cases for our application required for a Relational Database implementation. Hence, selection of MySQL would improve the development velocity.

In our machine learning service, we utilized LangChain with Flask to run Llama2 locally to minimize the costs. However, the response times were averaging 20 seconds due to the lack of GPU on our server. While we tried to maximize the performance using various methods such as quantization and multithreading, we could not meet our expected results. As a result, we decided to utilize Google's GenAI, Gemini (formerly Bard) which had a free web API. With the help of Gemini, we eliminated the need for hosting a LLM model in our local server. Also, with the growing need for a better infrastructure, we moved to Django from Flask to have a more stable experience and a higher development velocity.

Architecture Implementation & Development Process

On the backend, we adopted a microservice architecture to handle various functions: machine learning processes, a dictionary service, user management, and an API Gateway to integrate these services. We integrated our system with AWS, employing services like AWS API Gateway, Polly to generate human-like voices, Transcribe to transcribe speech to text, S3 to store audio files and profile images, Lambda to use the AWS clients, SNS to send push notifications, DynamoDB to save user device information, and Google Firebase Cloud Messaging for push notifications again.

To make local development smoother, we utilized Docker with Docker Compose on each of our microservices. This setup allowed us to maintain two primary branches on GitHub: 'dev' for development and 'main' for production, which are hosted on a remote Linux Ubuntu VPS server. After developing features locally, we opened pull requests (PRs) on GitHub. These PRs were merged only after a successful CI build and approval. Typically, PRs received feedback, prompting further iterations to refine the features or write cleaner code.

We use Github Actions, Docker, Docker Compose and DockerHub for CI/CD purposes. Also, we utilize MonitorRobot to track the status of our services and raise alerts for downed services. We leverage NGINX Amplify to monitor server health and statistics.

Our mobile application was developed using React Native. Which let us create an application that works seamlessly on both iOS and Android platforms. We made platform-specific adjustments as necessary to ensure compatibility. We have structured the React Native project in a modularized way

Brainstorming During Development

Regarding implementation details such as quiz features and gamification elements, we held meetings to discuss and plan these components. This collaborative approach helped us align on the vision and implementation strategies. Even though it meant a longer development process, thanks to these meetings we were in touch regarding the implementation of each feature and we were in track of what each of us are doing.

Brainstorming After Development

Each feature of our application went through multiple iterations. This iterative process allowed us to continuously improve the application based on user feedback and our own evaluations during the development cycle. For example, the “Active Words” functionality was not understood by most of the users during the GE250 test sessions. Some of the users didn’t even encounter it. After receiving those feedbacks, we decided to change the UI of the chat screen to display active words on top of the screen in a visually appealing way. We also planned other ways to signify active words, such as highlighting them in the chat. Lastly, in the next iterations, we would like to let the users determine the active words one by one.

5 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	13 May 2024-PASSED				

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	13 May 2024-PASSED, the chatbot responds with an average of 4 seconds				

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	13 May 2024-FAILED - The deepest feature is 4 levels nested (friend profile page accessed), but we are going to solve this issue by moving the friends screen to "Social" tab along with the leaderboard				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED, login streak is only updated on the initial login of the day.				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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 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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-FAILED, however, we are actively developing this feature by utilizing AWS SNS and it will be up and running very soon				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

Test ID	27	Category	Functional	Severity	Minor
Objective	This test case verifies that the statistics displayed in the Linguist 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	13 May 2024-PASSED				

Test ID	28	Category	Functional	Severity	Minor
Objective	This test case verifies that the statistics in the Linguist 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	13 May 2024-PASSED, statistics are displayed in the home screen for learning level of words

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	13 May 2024-PASSED, docker containers do restart upon failure				

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	13 May 2024-PASSED				

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 Linguist 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	13 May 2024-PASSED

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 Linguist 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	13 May 2024-PASSED, we integrated AWS Polly for this purpose which can generate human-like voice for texts.				

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	13 May 2024-PASSED, but sometimes there can be multiple correct choices. However, this issue arises rarely.				

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	13 May 2024-PASSED

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	13 May 2024-PASSED				

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	<p>Step 1. The user opens the app and starts a conversation with the chatbot.</p> <p>Step 2. The user enters text in a language other than English or random characters.</p> <p>Step 3. Repeat step 2 for five different languages.</p>
Expected	Chatbot responds in English, with a text saying it does not understand the message.
Date-Result	13 May 2024-PASSED, the bot responds in English to all sorts of inputs, ranging from non-English language (where it says it is an English language learning companion etc.) to emojis.

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	<p>Step 1. The user opens the app and initiates a conversation with the chatbot.</p> <p>Step 2. The user deliberately inputs hate speech, sexual content, or harassing messages into the chat.</p>				
Expected	The chatbot recognizes and addresses the inappropriate content by responding with a message that discourages such behavior and promotes respectful interaction.				
Date-Result	13 May 2024-PASSED, the bot says that it cannot respond to such messages.				

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	<p>Step 1. The user opens the app and starts a conversation.</p> <p>Step 2. The user taps on a word.</p> <p>Step 3. The user adds this word to a word list.</p> <p>Step 4. The user navigates to the word list screen.</p> <p>Step 5. The user taps on the relevant word list.</p>				
Expected	The word list displays the word added from the conversation screen.				
Date-Result	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED, when an unknown word is used "correctly", its confidence increases and is visible in the word lists page.				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-FALIED as this feature was removed since we decided that favorite lists do not have a purpose. Currently, lists can be activated and pinned, and both of these actions work smoothly.				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED				

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	13 May 2024-FAILED - This functionality has not been implemented				

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	13 May 2024-PASSED				

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	13 May 2024-PASSED

6 Maintenance Plan and Details

We have an existing feature road map planned and we are pursuing that plan. After completing the features on our road map, we want to stabilize the current codebase, and fix existing and arising bugs. Moreover, after “finishing” the features, before moving on with a new feature roadmap, we plan to iterate over the existing features to perfect them.

Currently we finished implementing our “Conversation & Chatting Engines” that includes chatting with multiple bots which extracts the user’s profile during a chat session and tries to teach the user active words chosen for that specific conversation. Along with the “Conversation & Chatting Engines” we have mechanisms to display word definitions with example sentences as well as a Word List mechanism so that the users can save words they want to use and select the word lists that they want the chatbot to use. We also finished several gamification elements such as statistics, chat streak, daily quests, leaderboard, friendship system. We have also started implementing the store page after the feedback we received during our GE250 testing sessions. All of the mentioned features will require further iterations to bring them to an even better place.

Our first and foremost priority will be continuing to improve the chatting experience and improve the chatting capabilities of our chatbots. In order to achieve this, we plan to fine tune a GPT-4 model so that it will focus more on the language teaching side than just being a chatbot. Also, by fine tuning we will be able to generate our multiple choice questions in a more correct way. We will also adjust the chatting UI to make it more user friendly and more understandable. The second priority will be increasing the gamification elements and finishing the implementation of in-app store screens. Third priority will be keeping the codebase scalable and clean.

Furthermore, we are currently running all services of a hosted VPS server in a single region. As we grow and focus on specific markets, we will move onto distributing and load-balancing our customers based on regions. We will utilize self-hosted VPS servers to keep our operating costs low for services besides GenAI and Machine Learning.

7 Other Project Elements

This section of the project analyzes various aspects of the engineering process beyond the core technical design and implementation. It emphasizes the integration of ethical practices, collaborative efforts, and the continuous learning and application of new knowledge throughout the project lifecycle.

7.1 Consideration of Various Factors in Engineering Design

When designing an engineering solution, multiple factors must be carefully balanced to ensure functionality, safety, and efficiency. Consideration of these factors is crucial in navigating the complex interplay of technical requirements, user needs, and environmental impacts. Therefore we will share our values within the context of constraints and standards of Linguist.

7.1.1 Constraints

We identified our constraints for Linguist to be Public Health, Public Safety, Global Factors, Cultural Factors, Social Factors, Environmental Factors and Economical Factors. We will analyze our values and provide our approach to these constraints.

7.1.1.1 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.

7.1.1.2 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.

7.1.1.3 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.

7.1.1.4 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.

7.1.1.5 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.

7.1.1.6 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 of 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.

7.1.1.7 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.

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.

7.1.2 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 Scrum guide (by Scrum.org), and our development team adheres to 4 to 1 week long Scrum cycles based on our schedules, 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.

7.2 Ethics and Professional Responsibilities

In developing Linguist, it is crucial to adhere to the highest ethical standards and professional responsibilities, especially given the app's role in education and personal development. Firstly, the use of AI and personalized learning algorithms must respect user privacy and data security. Ensuring that personal information and learning data are securely handled and only used to enhance the user experience is fundamental. Secondly, the customization features of Linguist must be designed to be inclusive, catering to a diverse range of learning needs and styles without discrimination. This includes accommodating users with disabilities and providing equal opportunities for learners from various backgrounds. Additionally, the AI-driven interaction must be transparent in its functioning, clarifying to users how their data influences the learning paths suggested by the app. This transparency builds trust and helps users feel more in control of their learning experience. Ethical considerations also extend to the content provided by the app, which should be culturally sensitive and unbiased to foster a respectful and supportive learning environment. Finally, professional responsibilities involve continuous monitoring and updating of the app to ensure it delivers effective and up-to-date educational content, maintaining its educational value.

We have prepared a privacy policy for the use of our application, which is seen in the application listings in the App Store and Play Store, and also a mandatory opt-in option during registration. The privacy policy can be seen via: <https://linguistai.app/privacy>.

7.2.1 Risks and Alternatives

We have identified the potential risks and alternatives for our application, in Tables 2 and 3 given below.

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 might get more popular.	Upon extensive discussion sessions, we can decide to implement chatbots having various personalities.
Users might want to use Linguist 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 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.

7.3 Teamwork Details

During the development of the project, we have utilized various tools and methods to strengthen the teamwork of our team. Our biggest achievements were a result of our strong teamwork and we will discuss our methods here.

7.3.1 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 4 to 1 week long Scrum cycles based on our schedules, 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. During the sprints, we continually monitored sprint metrics and charts (like burnup charts, see Figure 24 for an example) and took action/modified our sprint scope/goals accordingly.

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.

Workflow

Owned by Yağız Can Aslan
Last updated: Oct 20, 2023 • 1 min read

Please follow the workflow below when working on the project:

1. Make sure your task has a **valid Jira Task ID**.
2. **Create a new branch** ([How to get started working on a task using GitHub](#)).
 - a. **While working on your branch, please follow** [Commit Conventions](#).
3. After your work on your branch is completed, **open a Pull Request from your branch to the *dev* branch by following** [How to finish working on a task using GitHub](#) .
 - a. **While opening a Pull Request, please follow** [Pull Request Conventions](#).

Thank you for following our workflow! 🎉

Figure 11. Linguist Workflow

Pull Request Conventions before posting.'"/>

How to finish working on a task using GitHub

Owned by Yağız Can Aslan
Last updated: Oct 20, 2023 • 1 min read

When you finish working on a task and commit your changes to your respective branch, you can open a **"Pull Request (PR)"**, where you can request your branch **"LINGUIST-9999"** to be merged into the branch **"dev"**, and request code review(s).

Once the reviewer(s) approve your PR, your branch can be successfully merged into the **"dev"** branch (if no merge conflict(s) exist).

Please make sure your Pull Request follows [Pull Request Conventions](#) before posting.

Figure 12. Linguist Task Finishing Instructions

How to create and merge branches

1. Create your branch from the **dev** branch, as stated in our [getting started on a task Confluence guide](#).
2. When your work is done, merge back to the **dev** branch, as stated in our [finishing work on a task Confluence guide](#).
3. After a few merges to the **dev** branch, open a PR to the **test** branch. The **test** branch is utilized for collectively testing the interactions of a variety of features before merging to the **main** branch.
4. When tests are successful, open a PR to the **main** branch from the **test** branch.

dev branch: the branch which we work from.

test branch: the branch where we merge our changes and new features to test.

main branch: the branch which has the tested and working features.

Figure 13. Linguist Branch Instructions

How to get started working on a task using GitHub



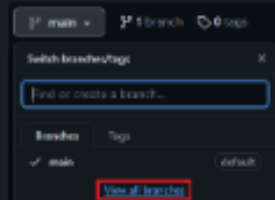
Owned by Yagiz Can Aslan · ...

Last updated: Oct 20, 2023 · 2 min read

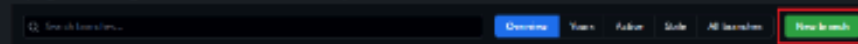
This short document demonstrates how to get started working on a task using new branches in GitHub.

Instructions

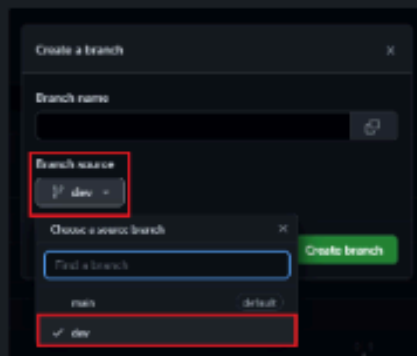
1. Go to our project's GitHub repository, locate and click "View all branches".



2. A page showing all active branches will open. Click "New branch".



3. Select the branch source as "dev".



4. Name your branch with the name of your Jira Task ID (example: **LINGUIST-999**, **NOT ERASMUS-999**) and click "Create branch".



5. Congratulations! You can now get started working on your task from the new branch 🎉



This document only covers how to get started working on a task using GitHub. The document for how to proceed upon task completion is here: [🔗 How to finish working on a task using GitHub](#).

Figure 14. Linguist Branch Creation Instructions

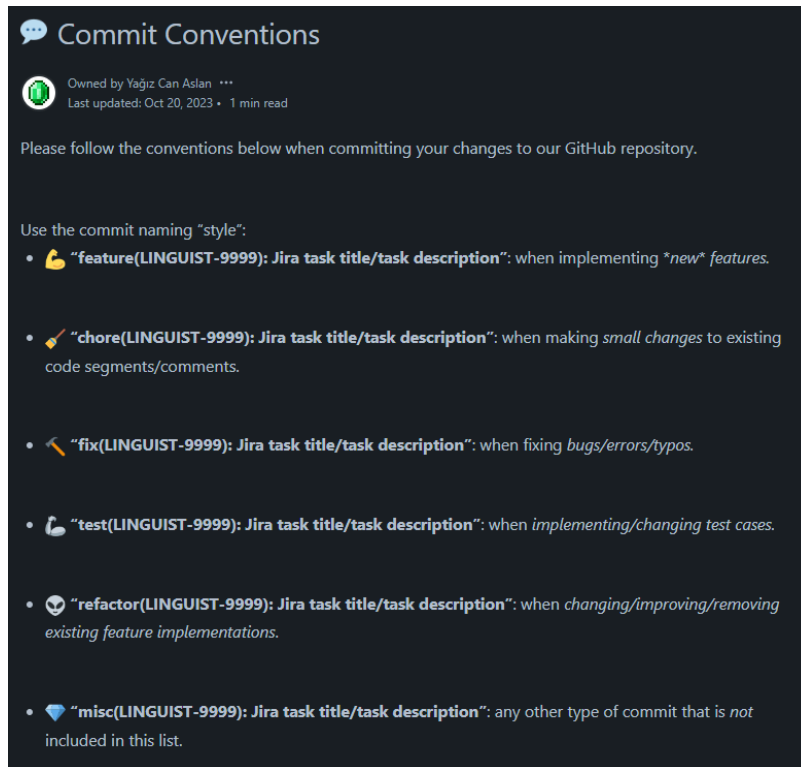


Figure 15. Linguist Commit Conventions

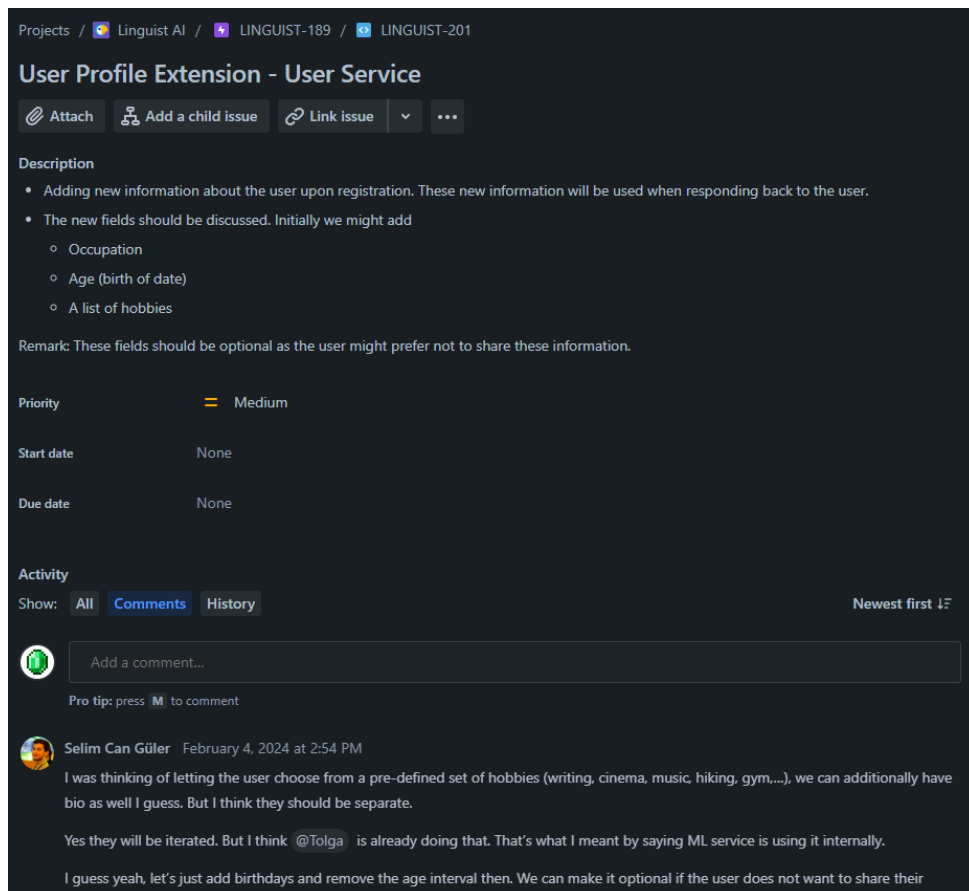



Figure 16. Example Jira Task



Pull Request Conventions

 Owned by Yağız Can Aslan
Last updated: Oct 20, 2023 • 1 min read

Please make sure your Pull Request (PR) includes the following sections (which will be provided by a file called "pull_request_template.md"):

- **Description:** A short description of the PR, and changes made.
- **Checklists:** This is just a header, no explanation is needed here.
 - **Development and Testing**
 - Related unit test(s) have been implemented and/or updated: Yes/No
 - Application changes have been tested: Yes/No
 - Related documents have been updated (if any): Yes/No
 - **Security**
 - Security impact and vulnerabilities of the change(s) have been considered: Yes/No
 - **Time and Space Complexity**
 - Time complexities of algorithms are not above $O(N^3)^*$ and are considered carefully: Yes/No
 - Space complexities of algorithms are considered carefully: Yes/No
 - **Code Review**
 - PR has a valid Jira Task ID: Yes/No
 - PR title follows [Commit Conventions](#), and is succeeded by a short explanation: Yes/No

feat(ERASMUS-87): Re-organize the project structure for types and tests.

 Merged can-aslan merged 5 commits into dev from ERASMUS-87 17 days ago

Figure 1: Example PR Title


 Make sure task id starts with "LINGUIST", not "ERASMUS".

Figure 17. Linguist Pull Request Conventions

LINGUIST-118 Bottom Navigation Icons Are Not Clickable	MISCELLANEOUS BUGS	TO DO		
LINGUIST-121 Determine LLM evaluation metrics and dataset	CONVERSATIONAL AI	TO DO		KC
LINGUIST-122 Research LLM deployment library	CONVERSATIONAL AI	TO DO		KC
LINGUIST-123 API Gateway Security Research	MICROSERVICE ARCHIT...	IN PROGRESS		
LINGUIST-125 Configure API Gateway	MICROSERVICE ARCHIT...	IN PROGRESS		
LINGUIST-124 Fork Dictionary API and Examine How It Works	WORD DICTIONARY	TESTING		
LINGUIST-128 Unknown Word Bank Backend Development	UNKNOWN WORD BANK	ON HOLD		
LINGUIST-134 Requirements Report - Introduction	REQUIREMENTS REPORT	TO DO	08 DEC	
LINGUIST-135 Requirements Report - Current System	REQUIREMENTS REPORT	TO DO	08 DEC	
LINGUIST-136 Requirements Report - Proposed System	REQUIREMENTS REPORT	TO DO	08 DEC	
LINGUIST-137 Requirements Report - Other Analysis Elements	REQUIREMENTS REPORT	TO DO	08 DEC	
LINGUIST-141 Register Does Not Validate Email and Password Strength	MISCELLANEOUS BUGS	TO DO		
LINGUIST-142 Design Conversation UML Diagrams	REQUIREMENTS REPORT	IN PROGRESS		
LINGUIST-165 Design Gamification (Streak, Leaderboard, Friends) UML Diagrams	REQUIREMENTS REPORT	TO DO		
LINGUIST-174 Examine Different Unknown Word Mapping Solutions	UNKNOWN WORD BANK	TO DO		
LINGUIST-175 Remove Unnecessary userId Field in UserStreak	GAMIFICATION V1	TO DO		
LINGUIST-176 Organize Jira Tasks and Report Title Formatting for Requirements Report	REQUIREMENTS REPORT	DONE		

Figure 18. Example Jira Sprint Backlog

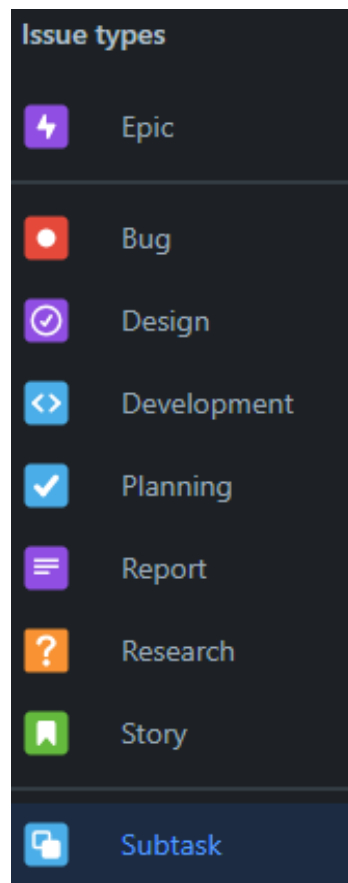


Figure 19. Jira Issue Types

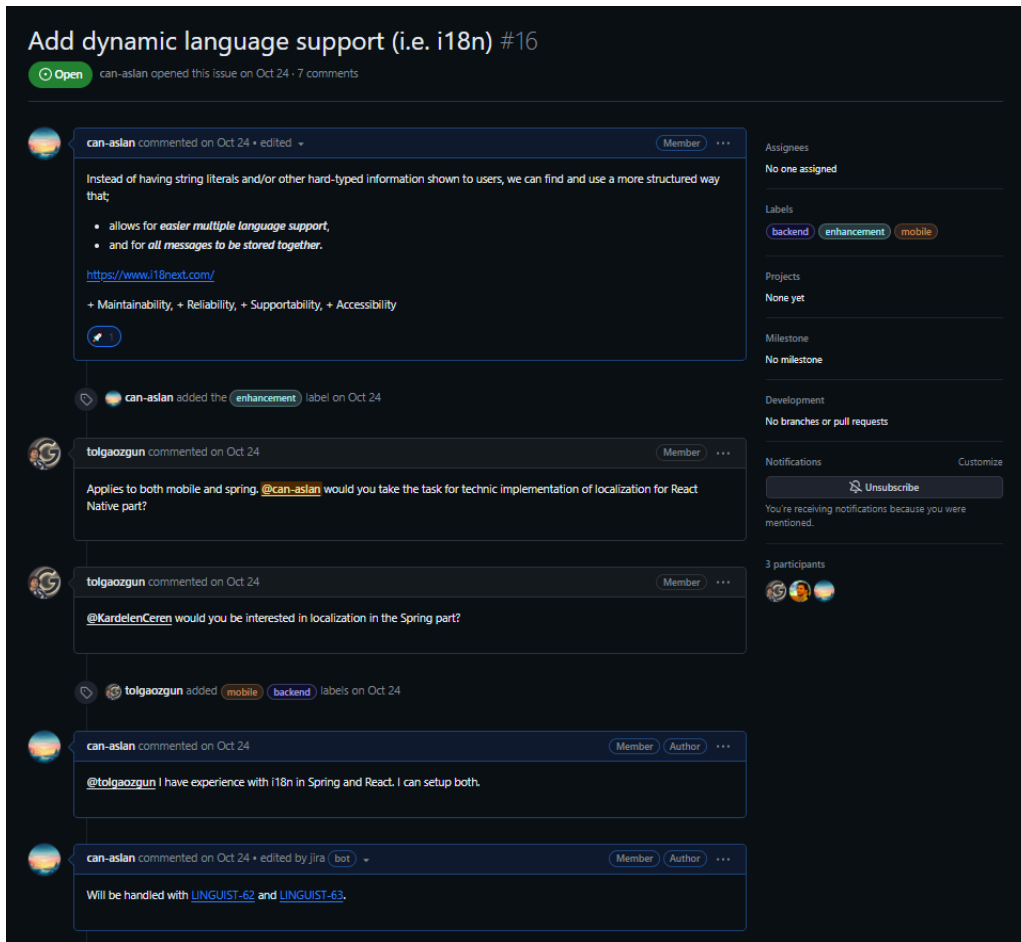


Figure 20. Example GitHub Issue Conversation

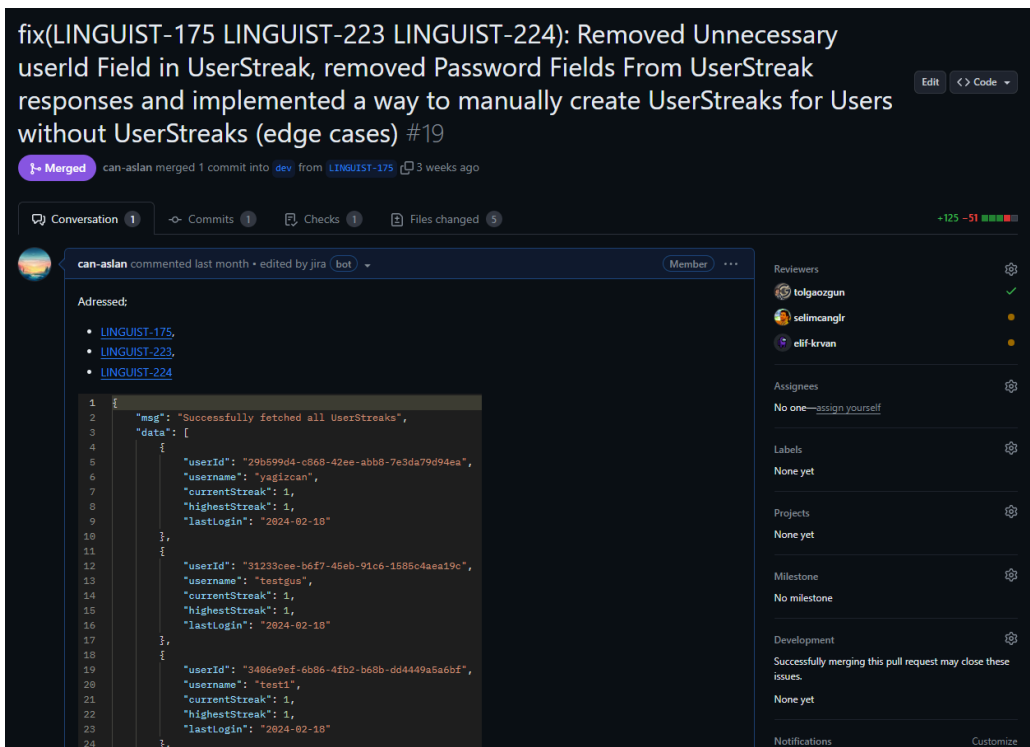


Figure 21. Example GitHub PR Title and Description

Merged

feat(LINGUIST-234): Added friendship #26
 elif-krvan merged 12 commits into `dev` from `LINGUIST-234` 2 weeks ago

```

14 +
15 + @Repository
16 + public interface IFriendshipRepository extends JpaRepository<Friendship, FriendshipId> {

```

can-aslan 3 weeks ago

Member ...

Indentation in this class seems off, can you check?

can-aslan 3 weeks ago

Member ...

i.e. some lines look 3 spaces

Reply...

Unresolve conversation

 elif-krvan marked this conversation as resolved.

src/main/java/app/linguistai/bmvp/service/gamification/FriendshipService.java

Hide resolved

```

22 + private final IAccountRepository accountRepository;
23 + private final IFriendshipRepository friendshipRepository;
24 +
25 + public Friendship sendFriendRequest(String user1Email, UUID user2Id) throws Exception {

```

can-aslan 3 weeks ago

Member ...

Can you make this method `@Transactional`

Reply...

Unresolve conversation

 elif-krvan marked this conversation as resolved.

src/main/java/app/linguistai/bmvp/service/gamification/FriendshipService.java
 Outdated

Hide resolved

```

12 + import app.linguistai.bmvp.model.User;
13 + import app.linguistai.bmvp.model.enums.FriendshipStatus;
14 + import app.linguistai.bmvp.repository.IAccountRepository;
15 + import jakarta.transaction.Transactional;

```

can-aslan 3 weeks ago

Member ...

Just fyi, using `org.springframework.transaction.annotation.Transactional` rather than `jakarta.transaction.Transactional` can be better for performance and other optimizations. So you can consider changing it but I think it is also okay if it stays like this as stated explained in the accepted answer [here \(StackOverflow\)](#)

Figure 22. Example GitHub PR Review

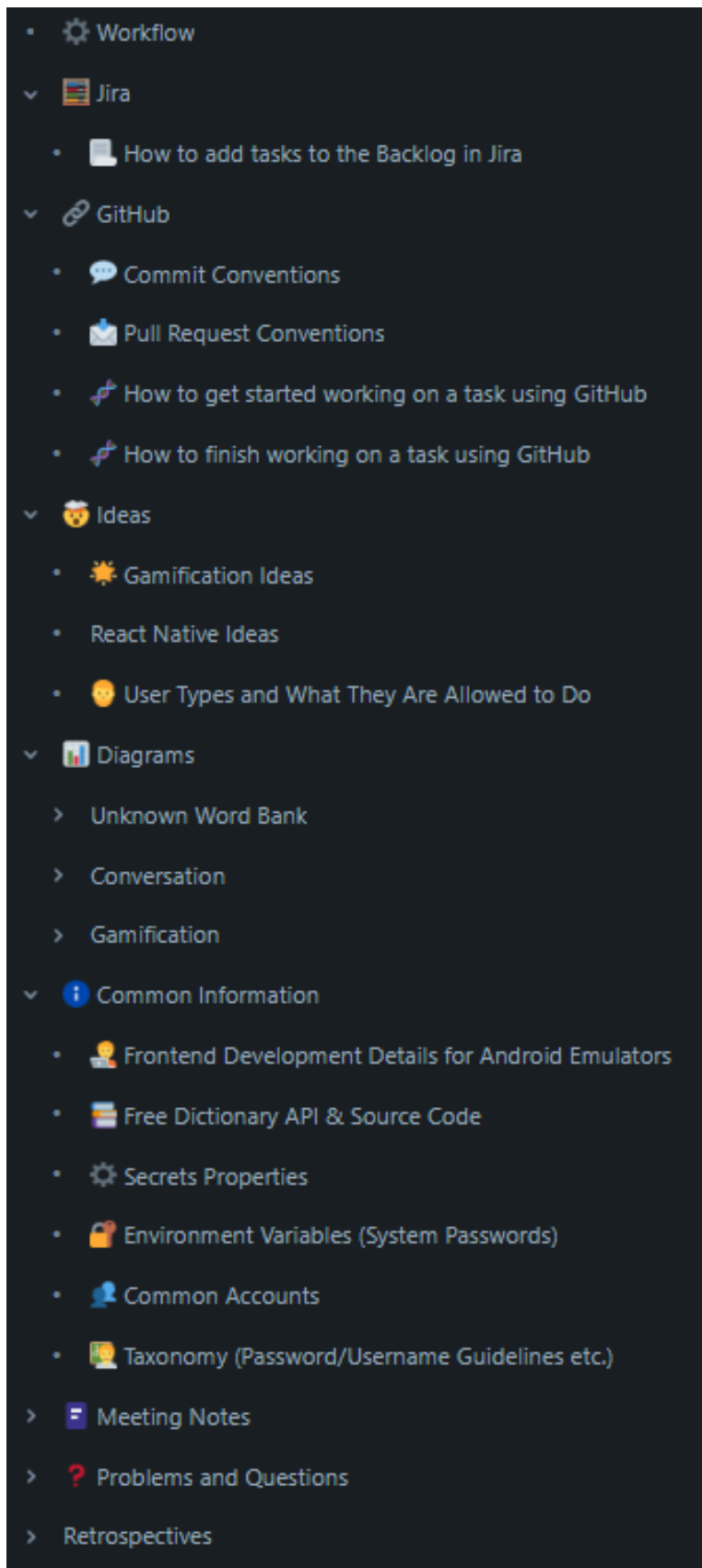
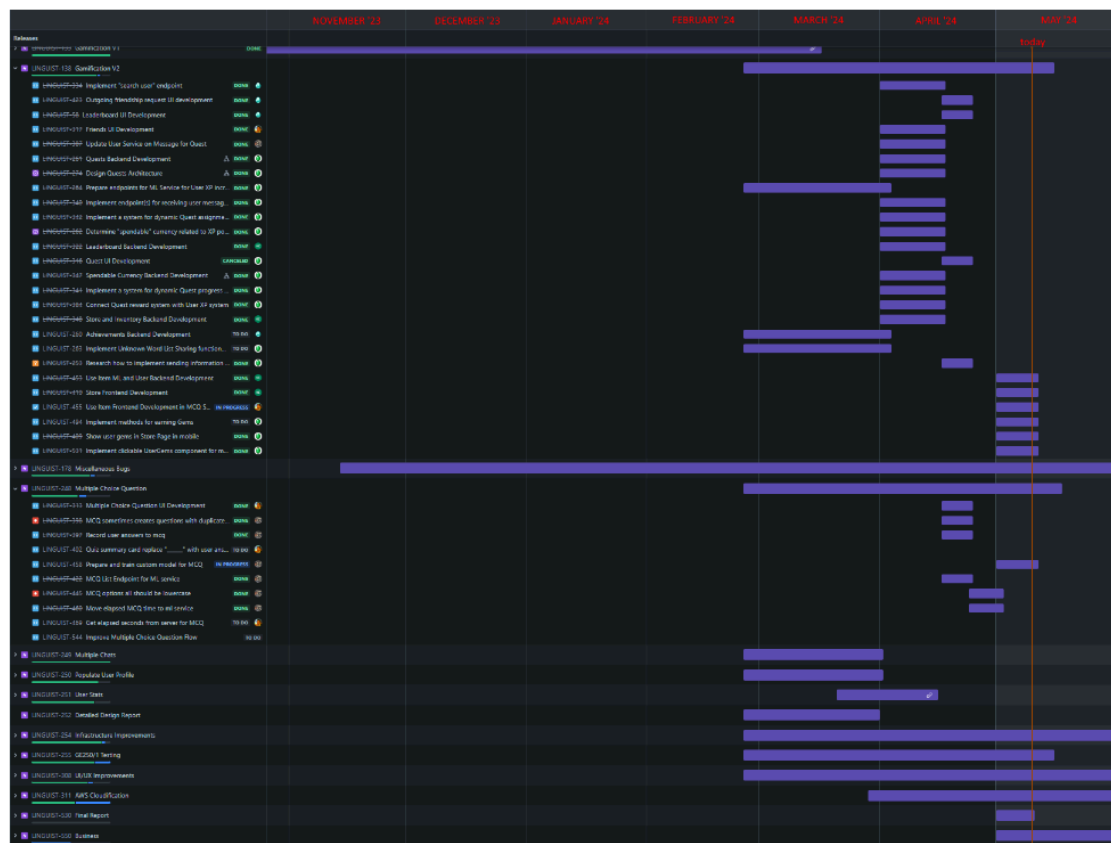
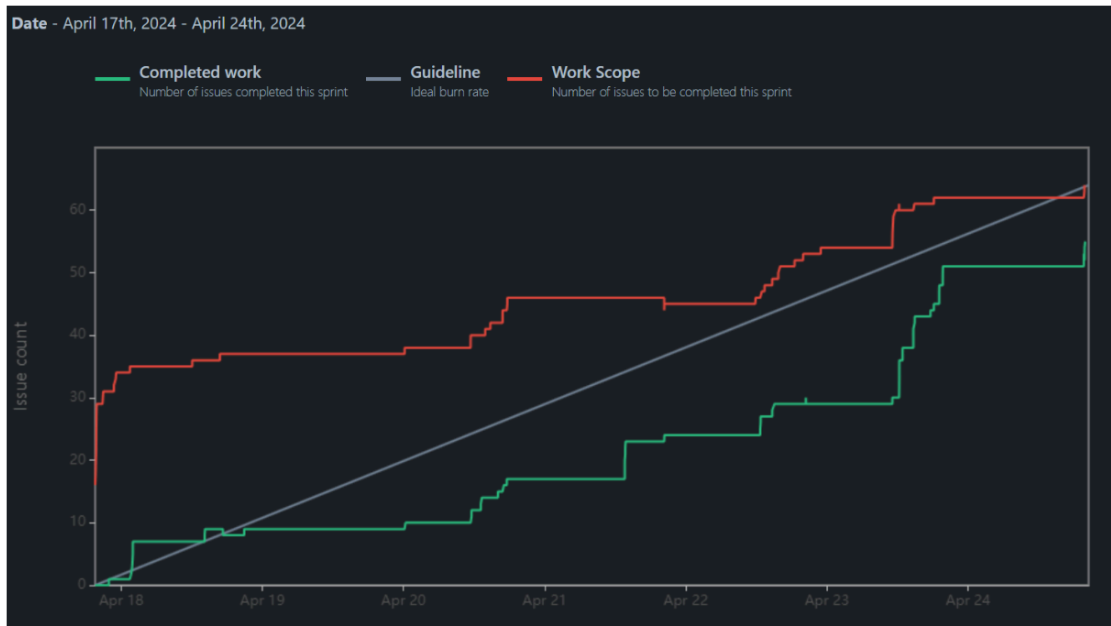


Figure 23. Confluence Pages Summary Overview



7.3.2 Helping creating a collaborative and inclusive environment

As stated in section 8.1, 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.

7.3.3 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 Co-Lead, Scrummaster, Frontend, AWS Integration
- Kardelen Ceren: Backend, Frontend, Machine Learning
- Selim Can Güler: Frontend Lead, Backend, AWS Integration
- İlkim Elif Kervan: Backend Co-Lead, Frontend, AWS Integration
- Tolga Özgün: Machine Learning Lead, DevOps, Infrastructure, Backend

7.3.4 Meeting objectives

This section will detail which work packages are completed (and/or at which level) that were set in the project plan as part of the Analysis report.

Table 4: List of initial work packages

WP#	Work package title	Leader	Members involved
WP1	Unknown Word Bank Backend Development	Yağız Can Aslan	İlkim Elif Kervan
WP2	Unknown Word Bank UI Development	Selim Can Güler	Tolga Özgün, Yağız Can Aslan
WP3	Inferring English Level - Research	Kardelen Ceren	Selim Can Güler, Tolga Özgün
WP4	Dictionary Service Development	İlkim Elif Kervan	Yağız Can Aslan, Selim Can Güler
WP5	Conversational Chatbot LLM Development	Tolga Özgün	Kardelen Ceren
WP6	Conversation Backend Infrastructure Development	Yağız Can Aslan	İlkim Elif Kervan
WP7	User Message Filter Development	Kardelen Ceren	Selim Can Güler
WP8	Chatbot Message Filter Development	Tolga Özgün	Kardelen Ceren
WP9	Contextual Word Scoring Model Development	Selim Can Güler	Tolga Özgün, Kardelen Ceren
WP10	Multi-choice Question Test Backend Development	İlkim Elif Kervan	Yağız Can Aslan
WP11	Multi-choice Question Test UI Development	Selim Can Güler	Tolga Özgün
WP12	Gamification Backend Development	Yağız Can Aslan	İlkim Elif Kervan
WP13	Gamification UI Development	Selim Can Güler	Yağız Can Aslan
WP14	API Gateway, Microservice Architecture and Authorization	İlkim Elif Kervan	Yağız Can Aslan
WP15	Setup Jira and Confluence Infrastructures	Yağız Can Aslan	-
WP16	Voice Chat Integration	Kardelen Ceren	İlkim Elif Kervan, Yağız Can Aslan
WP17	User Profile Backend Development	Tolga Özgün	Kardelen Ceren, İlkin Elif Kervan

WP 1: Unknown Word Bank Backend Development			
Start date: 14.11.2023 End date: 30.12.2023			
Leader:	Yağız Can Aslan	Members involved:	İlkim Elif Kervan
Objectives: Designing the backend of the “Unknown Word Bank” system. Development of object models, database schemas and logic of the “Unknown Word Bank” system including adding and removing words from word lists, creation and deletion of word lists, activating and deactivating lists.			
Tasks:			
Task 1.1 UML Use Case Diagram for Unknown Word Bank: Design, discuss and iterate over the UML Use Case diagram for Unknown Word Bank system.			
Task 1.2 UML Activity Diagram for Unknown Word Bank: Design, discuss and iterate over the UML Activity diagram for Unknown Word Bank system.			
Task 1.3 UML State Diagram for Unknown Word Bank: Design, discuss and iterate over the UML State diagram for Unknown Word Bank system.			
Task 1.3 UML Class Diagram for Unknown Word Bank: Design, discuss and iterate over the UML Class diagram for Unknown Word Bank system.			
Task 1.4 Unknown Word Bank Backend Spring Implementation: Implement the Controller - Service - Repository classes in Java Spring, modeled object classes and database schemas for the Unknown Word Bank system.			
Deliverables			
D1.1: Source Code			
D1.2: Documentation			
D1.3: UML Use Case, Activity, State and Class Diagrams			
100% COMPLETED			

WP 2: Unknown Word Bank UI Development			
Start date: 10.12.2023 End date: 15.01.2024			
Leader:	Selim Can Güler	Members involved:	Tolga Özgün, Yağız Can Aslan
Objectives: Designing the UI of the “Unknown Word Bank” system. Development of UI and logic of the “Unknown Word Bank” system including adding and removing words from word lists, creation and deletion of word lists. Sending calls to “Unknown Word Bank” endpoints when necessary			
Tasks:			
Task 2.1 Designing the UI : Designing the general look of the UI.			
Task 2.2 Development of UI : Implementing the design in React Native without any logic included.			
Task 2.3 Development of logic/state: Logical components, state changes in the frontend must be implemented.			
Task 2.4 Integrating API endpoint calls: Endpoints for adding and removing unknown words to word lists, creation and deletion of word lists must be sent to the backend.			
Deliverables			
D2.1: UI Design			
D2.2: Source Code			
D2.3: Documentation			
100% COMPLETED			

WP 3: Inferring English Level - Research			
Start date: 01.01.2024 End date: 30.01.2024			
Leader:	Kardelen Ceren	Members involved:	Selim Can Güler, Tolga Özgün
Objectives: Researching methods to correctly and effectively assess a learner's English level in order for the LLM to be able to match users level. As this is an internal measure, evaluated levels do not necessarily have to match CEFR levels such as B1, C2, etc., but would be preferred if possible.			
Tasks: Task 3.1 Literature review: Review comprehensively the existing research on English evaluation tests, their accuracy and how well they could be incorporated to an online, automated system. Task 3.2 Market research: Research the existing online tools that are used to evaluate foreign language skills. Task 3.3 Consulting language learning experts: Conduct interviews with language learning professionals to get insight into different benchmarks and their recommendations for our system. Task 3.4 Development of evaluation criteria: Define clear criteria for language evaluation that is feasible and as accurate as possible.			
Deliverables D3.1: Research Report D3.2: Evaluation Criteria Guidelines			
100% COMPLETED			

WP 4: Dictionary Service Development			
Start date: 14.11.2023 End date: 15.0.2023			
Leader:	İlkin Elif Kervan	Members involved:	Yağız Can Aslan, Selim Can Güler
Objectives: In this work package, a dictionary service will be developed in order to provide word definitions and their usages in the sentences to the user. Dictionary API will not be created from scratch. A free dictionary API will be selected and integrated into the project. Since it is easier to send requests to external APIs with Express, a new microservice will be implemented using NodeJS and Express. The sole purpose of this microservice will be to send request to external APIs whenever needed.			
Tasks: Task 4.1 Research about available dictionary APIs : Make a research about the existing free dictionary APIs on the Internet to find a suitable one to use in the project. Task 4.2 Integrate dictionary API to the project : In order to display vocabulary definition and usage to the user, implement a microservice to send requests to the dictionary API found in the first subtask.			
Deliverables D4.1: An Express microservice D4.2: Source code of the API D4.3: Documentation of the API			
100% COMPLETED			

WP 5: Conversational Chatbot LLM Development			
Start date: 05.11.2023 End date: 20.01.2024			
Leader:	Tolga Özgün	Members involved:	Kardelen Ceren
<p>Objectives: In this work package, conversational chatbot LLM will be developed and deployed in a server that will handle the user input to create a conversation. During this development various experiments with existing chat models will be conducted to select the best performing model for our use case. Moreover, we will try to leverage methodologies such as prompt engineering, LLM chaining and agents to create the most efficient chatbot.</p> <p>Tasks:</p> <p>Task 5.1 Research and select a framework on chat models: There are many frameworks that help with the deployment and management of chat models. With the use of a framework, we can focus on the details of our application without having to worry about the basics.</p> <p>Task 5.2 Research and select chat models : We can leverage more than one chat model in our application as a chain, however we need to research and calculate the feasibility for each combination.</p> <p>Task 5.3 Prompt engineering : Experiment with various prompts in various use cases to find the best performing prompt.</p> <p>Task 5.4 Research and ingest documents : We can ingest documents to our chatbot which puts additional knowledge into the chatbot's memory. This can be beneficial for our use case as it decreases the need for fine-tuning the model.</p> <p>Task 5.5 Research and create tools: There are tools that can be defined to the chatbot to help with the conversation. We can research and see if the implementation is feasible.</p>			
<p>Deliverables</p> <p>D5.1: Source code of LLM microservice</p> <p>D5.2: Various LLM API endpoints</p> <p>D5.3: Documentation of the API</p>			
100% COMPLETED			

WP 6: Conversation Backend Infrastructure Development			
Start date: 22.10.2023 End date: 13.11.2023			
Leader:	Yağız Can Aslan	Members involved:	İlkin Elif Kervan
<p>Objectives: The user needs to be able to send and receive messages from our system. This work package aims at providing the infrastructure for storing (not producing) the messages. There needs to be support for multiple conversations.</p> <p>Tasks:</p> <p>Task 6.1 Design Conversation ER Diagram: Design the ER diagram for the Conversation system which will aid in the designing of database schemas.</p> <p>Task 6.2 Conversation Backend Spring Implementation : Implement the Controller - Service - Repository classes in Java Spring, modeled object classes and database schemas for the Conversation system.</p>			
<p>Deliverables</p> <p>D6.1: Source Code</p> <p>D6.2: Documentation</p> <p>D6.3: Conversation ER Diagram</p>			
100% COMPLETED			

WP 7: User Message Filter Development			
Start date: 30.01.2024 End date: 30.02.2024			
Leader:	Kardelen Ceren	Members involved:	Selim Can Güler
Objectives: <i>This work package aims to filter user's messages so that texts that include sexist, racist, discriminating, hateful or sexual phrases, a warning message can be sent instead of an actual bot response.</i>			
Tasks: Task 7.1 Determine malicious words and phrases : <i>Research coding libraries or APIs that have listed unsuitable English vocabulary.</i> Task 7.2 Develop user text filter : <i>Implement a user filter that checks for malicious intent and sends a warning message, blocking the text reaching the bot.</i>			
Deliverables D7.1: Source code D7.2: Documentation			
100% COMPLETED			

WP 8: Chatbot Message Filter Development			
Start date: 20.01.2024 End date: 30.02.2024			
Leader:	Tolga Özgün	Members involved:	Kardelen Ceren
Objectives: <i>Chatbot's messages must be filtered to not demonstrate any racism, sexism, discriminatory behavior, hate crime, sexual behavior or hallucinatory information.</i>			
Tasks: Task 8.1 Literature and tool review: <i>Research existing literature and LLM filtering tools to determine which phrases could and should be filtered, along with which tools can be used to accomplish the development.</i> Task 8.2 Filter implementation : <i>Implement a filter that checks the text that comes out of the LLM based on the malicious words and phrases determined in Task 8.1.</i>			
Deliverables D8.1: LLM filtering guidelines report D8.2: Source code D8.3: Documentation			
100% COMPLETED			

WP 9: Contextual Word Scoring Model Development			
Start date: 15.01.2024 End date: 30.01.2024			
Leader:	Selim Can Güler	Members involved:	Tolga Özgün, Kardelen Ceren
Objectives: <i>There needs to be a model that will implicitly score the unknown words used by the user based on contextual and semantic correctness. The model will be an LLM. It will be fine-tuned for the task.</i>			
Tasks: Task 9.1 Development & fine-tuning of the model : Development and testing of the initial model, and fine-tuning it to increase its performance. Task 9.2 Updating the database: Update the user's score for the words affected Task 9.3 Develop API: Development of several API endpoints related to the scoring of words in a sentence.			
Deliverables D9.1: Feasibility Report D9.2: Code D9.3: Documentation D9.4: LLM Model			
100% COMPLETED			

WP 10: Multi-choice Question Test Backend Development			
Start date: 01.03.2024 End date: 15.04.2024			
Leader:	İlkim Elif Kervan	Members involved:	Yağız Can Aslan
Objectives: <i>In this work package, backend services of the multi-choice tests will be implemented. The purpose of these tests is to measure the success of the users after they have conversation with the bot. In order to assess user success we will test them before and after the conversation. The tests before and after the conversation will test the same words using different questions. Therefore, in this package we will implement services that prepare questions and assess user success.</i>			
Tasks: Task 10.1 Make a research to find the most effective question types : Identification of the best question type so that success of the user can be measured correctly. Task 10.2 Generate test questions : Development of the question generator services using generative AI. Task 10.3 Assess success of the user : Development of services to measure the success of the user before and after they have conversation with the chat bot.			
Deliverables D10.1: Source code. D10.2: Documentation D10.3: LLM Model			
100% WORKING - We are iterating on this WP based on the feedback we have acquired from our GE250/1 testing sessions and our own testing			

WP 11: Multi-choice Question Test UI Development			
Start date: 01.03.2024 End date: 15.04.2024			
Leader:	Selim Can Güler	Members involved:	Tolga Özgün
Objectives: Multi-choice question test will appear once the user starts a conversation with the bot and once the conversation ends. It will include calling the backend for the generation of questions, displaying the questions in an aesthetic manner. When the test ends results will be displayed.			
Tasks: Task 11.1 Test question UI development: The UI will include the following components: displaying questions and ability to go back and forth between questions, feedback on wrong/correct questions, displaying results summary after test ends Task 11.2 Integration with backend : Includes sending the answers to backend, fetching the questions from the backend.			
Deliverables D11.1: UI Design D11.2: Code D11.3: Documentation			
100% WORKING - We are iterating on this WP based on the feedback we have acquired from our GE250/1 testing sessions and our own testing.			

WP 12: Gamification Backend Development			
Start date: 22.10.2023 End date: 30.04.2024			
Leader:	Yağız Can Aslan	Members involved:	İlkim Elif Kervan
<p>Objectives: Gamification will be a vital part of our application. It will distinguish us from competitors while aiding our users' language learning journeys. Interesting and captivating gamification ideas need to be implemented such as; login/chat streaks, experience points, customization, leaderboards, friendship system, achievements, daily quests and more (as ideas come up and are tested).</p>			
<p>Tasks:</p> <p>Task 12.1 UML Use Case Diagram for Gamification: Design, discuss and iterate over the UML Use Case diagram for Gamification system.</p> <p>Task 12.2 UML Activity Diagram for Gamification: Design, discuss and iterate over the UML Activity diagram for Gamification system.</p> <p>Task 12.3 UML State Diagram for Gamification: Design, discuss and iterate over the UML State diagram for Gamification system.</p> <p>Task 12.3 UML Class Diagram for Gamification: Design, discuss and iterate over the UML Class diagram for Gamification system.</p> <p>Task 12.4 Login/Chat Streak Backend Implementation: Implement the Controller - Service - Repository classes in Java Spring, modeled object classes and database schemas for the Login/Chat Streak system.</p> <p>Task 12.5 Leaderboard Backend Implementation: Implement the Controller - Service - Repository classes in Java Spring, modeled object classes and database schemas for the Leaderboard system.</p> <p>Task 12.6 Friendship Backend Implementation: Implement the Controller - Service - Repository classes in Java Spring, modeled object classes and database schemas for the Friendship system.</p> <p>Task 12.7 Achievements and Daily Quests Backend Implementation: Implement the Controller - Service - Repository classes in Java Spring, modeled object classes and database schemas for the Achievements and Daily Quests system.</p>			
<p>Deliverables</p> <p>D12.1: Source Code</p> <p>D12.2: Documentation</p> <p>D12.3: UML Use Case, Activity, State and Class Diagrams</p>			
100% COMPLETED - Task 12.1, Task 12.2, Task 12.3, Task 12.3, Task 12.4, Task 12.5, Task 12.6			
50% COMPLETED - Task 12.7. Daily Quests are 100% completed, however, we have deprioritized achievements development and hence paused its development.			

WP 13: Gamification UI Development			
Start date: 20.11.2023 End date: 30.04.2024			
Leader:	Selim Can Güler	Members involved:	Yağız Can Aslan
<p>Objectives: Development of gamification elements. Chat streak components will be displayed on each initial load of the application. List of achievements will be visible from the user's profile (obtained and to be obtained), achievement badges will be displayed when their conditions are met. Leaderboard based on chat streak and learned words between user's friends and a randomly generated league will be displayed. Personification of the chatbot.</p>			
<p>Tasks:</p> <p>Task 13.1 Personification of the chatbot : Giving a custom persona to the chatbot including its name, profile image, general behaviors.</p> <p>Task 13.2 Listing accomplished achievements : Listing the predefined obtained and to be obtained achievements on the user's profile, on a separate tab.</p> <p>Task 13.3 Listing badges when conditions are met : There should be a notification when one of the achievements are accomplished. A badge will be given for each achievement.</p> <p>Task 13.4 Leaderboard for chat streak and learned words : A leaderboard where the user can see their ranking among their friends or in the league they are assigned to based on both their chat streak and the number of learned words.</p> <p>Task 13.5 Chat Streak: A chat streak component displaying the user's current and highest streak with an appealing visual will be developed. It will be displayed on each login. It must also be accessible from the home page.</p>			
<p>Deliverables</p> <p>D13.1: UI Design</p> <p>D13.2: Code</p> <p>D13.3: Documentation</p>			
100% COMPLETED - Task 13.1, Task 13.5			
INCOMPLETE - Task 13.2, Task 13.3 (achievements development are paused as stated in WP 12 status description), Task 13.4 (a global leaderboard and friend leaderboard exists, but for experience points only. no leaderboard exists as of yet for chat streaks and/or learnt words).			

WP 14: API Gateway, Microservice Architecture, Authentication and Authorization			
Start date: 22.10.2023 End date: 15.12.2023			
Leader:	İlkim Elif Kervan	Members involved:	Yağız Can Aslan
Objectives: <i>Implementing API Gateway and microservice architecture. Configuring API Gateway to direct the incoming requests to related services. Developing security for gateway and microservices. Engineering login and register services and integrating it with the database for authentication and authorization.</i>			
Tasks: Task 14.1 Create API Gateway : <i>Implementing API Gateway with Spring Boot and configuring it to redirect requests to related microservices.</i> Task 14.2 Implement security : <i>Developing login and register services using JWT. Implementing authorization and authentication layers in the API gateway and microservices to secure the endpoints from malicious requests.</i>			
Deliverables D14.1: <i>Spring Boot API Gateway</i> D14.2: <i>Authorization and authentication layers</i> D14.3: <i>Source code</i> D14.4: <i>Code documentation</i>			
100% COMPLETED			

WP 15: Setup Jira and Confluence Infrastructures			
Start date: 15.10.2023 End date: 22.10.2023			
Leader:	Yağız Can Aslan	Members involved:	-
Objectives: <i>Setup our issue tracking, version control and bug reporting infrastructures. Setup Jira workflow diagrams and issue types. Publish crucial Confluence workflow guidelines and category-based folders.</i>			
Tasks: Task 15.1 Setup Jira and Issue Types: <i>Setup our issue tracking, version control and bug reporting infrastructures. Setup Jira workflow diagrams and issue types.</i> Task 15.2 Setup Crucial Confluence Documents: <i>Publish crucial Confluence workflow guidelines and category-based folders.</i>			
Deliverables D15.1: <i>Jira URL</i> D15.2: <i>Jira Issue Types</i> D15.3: <i>Confluence URL</i> D15.4: <i>Multiple Confluence Documents & Guidelines</i>			
100% COMPLETED			

WP 16: Voice Chat Integration			
Start date: 20.02.2024 End date: 30.03.2024			
Leader:	Kardelen Ceren	Members involved:	İlkim Elif Kervan, Yağız Can Aslan
Objectives: Implementing speech-to-text for the user and text-to-speech for the chatbot. It is important to choose a voice that is as human-sounding as possible.			
Tasks: Task 16.1 Research speech-to-text and text-to-speech libraries : Research libraries and APIs that enable voice chat integration. Task 16.2 Implement speech-to-text : Integrate the speech-to-text libraries to transcribe user's speech to be fed into the main model that generates the response. Task 16.3 Implement text-to-speech : Integrate the text-to-speech libraries to voice chatbot's responses if desired. A human-like voice should be selected.			
Deliverables D16.1: Source code D16.2: Documentation			
100% COMPLETED - Task 16.1, Task 16.3 (we used AWS Polly for Text-to-Speech)			
100% WORKING - Task 16.2 (we used AWS Transcribe and it is functional, yet we are still polishing this feature)			

WP 17: User Profile Backend Development			
Start date: 20.02.2024 End date: 30.03.2024			
Leader:	Tolga Özgün	Members involved:	Kardelen Ceren, İlkim Elif Kervan
Objectives: Users' profile, i.e. their likes and dislikes and demographic information, will be kept in the database and frequently updated based on their messages. This profile will serve to generate messages from the chatbot that will interest the user to engage in long conversations.			
Tasks: Task 17.1 User Profile Generator LLM Development : Implement an LLM using prompt engineering that outputs the user profile when an old profile and user's latest messages are given. Task 17.2 User Profile Backend Development: Implement the Controller - Service - Repository classes in Java Spring, modeled object classes and database schemas for the user profile system by modifying the existing user model. Task 17.3 Research and Improve: Research to find better methodologies which may include leveraging agents, tools and chaining.			
Deliverables D17.1: Source code D17.2: Documentation D17.3: LLM			
100% COMPLETED			

7.4 New Knowledge Acquired and Applied

During the project, we have achieved to host a successful production-level microservice architecture with over 99% uptime. We have acquired real users to our application and were able to test various features and UI/UX during the development phase. As a consequence, we have gained knowledge in various areas.

- Developing multi-platform mobile application

None of the team members had previous experience in building a mobile application. After defining the purpose and the boundaries of our project, we decided on the main parts each of the members will work on. After that, those who would work directly in building the mobile application planned learning about mobile application development. We chose to use React Native to build the mobile application. For that purpose, we followed online learning methods, and during the project we took a Udemy course about using React Native for mobile application development.

- Designing a scalable architecture for our system

To make sure that we will have a scalable architecture, we decided that we needed to know more about the current architectural approaches and their details. For that purpose, we assigned team members to learn more about microservice architecture, network communication methods used in microservice architecture such as gRPC, HTTP, and messaging queues, and which is the most feasible one in terms of our structure. We learnt more about this by reading blog posts about software architectures from companies like Microsoft and reading more about them.

- Choosing the most suitable databases for our architecture

Since our application includes a chatting system with an LLM we conducted research on how similar systems utilize different database systems. We did an initial research on how Discord uses NoSQL databases by reading their analysis of different approaches. We iteratively learnt more about how large-scale systems such as Discord work by using online learning methods, and selected our databases for our architecture accordingly.

- Utilizing GenAI in various tasks

We have utilized multiple GenAI models such as Mistral, Llama 2, Llama 3, Gemini, GPT-3.5-turbo and GPT-4-turbo to perform our use-cases. We have measured these models in terms of their time-to-respond, time for first meaningful token, quality of message, ability to follow instructions, ability to be fine-tuned and API support. We have leveraged Prompt Engineering to utilize the models to their best performance. We are also in the process of fine-tuning the GPT-3.5-turbo model. Our use cases consist of scoring the use of words in sentences by the user, extracting user information from their chat history, and generating chat messages that utilize profile information, unknown words, and various bot profiles.

As GenAI solutions are new and in constant development, we have learned how to track the performance of different models, utilize LangChain, Ollama, Python and Web APIs to run the models. We have also learned how to fine-tune the models, and various model tuning methods such as LoRA, QLoRA and Adapter Tuning.

- Developing CI/CD pipeline for microservice architecture

We have API Gateway, User, Machine Learning and Dictionary services that are running in our Linux Ubuntu VPS Server that utilize CI/CD methods. We utilize Github Actions to build and deploy our code to the server. In order to maintain the architecture we also utilize Docker and Docker Compose so that we can eliminate any OS or machine-dependent issues. Before this project, we had not maintained a project that utilized a microservice architecture with CI/CD. We have learned how to create Github Workflow files, utilize Github Actions, Dockerhub, Docker Compose and Docker to deploy code into a live server.

- Learning more about language teaching in casual settings

We had a meeting with Özlem Doğan, who is an instructor of the Spanish courses in Bilkent. We discussed the process of language learning and how we can design our application in a way that the process is natural to the users so they don't feel like they are memorizing words. Also, we talked about the factors that can attract learners to keep using our app.

- Creating a unique visual language for the best user experience

We had an initial meeting with Jülide Akşiyote Görür, from the COMD department, about building a visual language for our application. We showed our UI to them after building a sample visual language of our own to learn more about how we can do better in that regard.

8 Conclusion and Future Work

8.1 Conclusion

In conclusion, Linguist represents a comprehensive and innovative solution for individuals learning English as a foreign language. By leveraging advanced language learning models and intuitive user interfaces, the application facilitates immersive and engaging language practice through text- or speech-based interactions with customizable chatbots. The incorporation of features such as real-time word definitions, vocabulary testing, and personalized conversations enhances user engagement and effectiveness.

Moreover, gamification elements add an element of fun and motivation to the learning process, encouraging users to stay committed and track their progress over time. From chat streaks and word learning statistics to leaderboards and daily quests, Linguist fosters a dynamic and supportive learning community where users can challenge themselves, connect with others, and celebrate their language learning achievements.

Finally, the application's ability to adapt and personalize learning experiences based on user preferences underscores its commitment to providing tailored language learning support. By continuously refining user profiles and chatbot responses, Linguist strives to deliver a truly immersive and personalized learning journey for each individual user. As the application continues to evolve and expand its offerings, it holds promise as a valuable tool for language learners worldwide.

8.2 Future Work

As Linguist continues to grow and evolve, there are several areas of focus for future development and enhancement. Based on the feedback we received during our user testing sessions, the following outlines key areas of consideration for expanding and refining the application:

- **Expansion to Other Languages:** Adding support for additional languages such as French, Spanish, German, Japanese, and more, to cater to a broader user base and facilitate language learning in multiple languages.
- **Enhanced Competition Features:** Introduce more competition types and interactive challenges for users to engage with friends, fostering a sense of community and healthy competition within the app.
- **Diversification of Chatbot Selection:** Expand the variety of chatbots available to users, each with unique personalities and teaching styles. Additionally, we're considering allowing users the option to customize their chatbot experience by selecting the bot's desired characteristics from a list.
- **Advanced Statistics and Insights:** Implement additional statistics tracking features, including past quiz performance and learning progress metrics, to provide users with detailed insights into their language learning journey and areas for improvement.
- **Expansion of In-App Store:** Introduce a wider range of in-app store items, including consumables, themes, and cosmetic items, to enhance user engagement and offer personalized customization options.
- **Accessibility Features:** Implement dark mode and themes optimized for colorblind users to improve accessibility and user experience for all users.

- **UI/UX Refinement:** Continuously refine the user interface and user experience, focusing on polishing visual elements, streamlining navigation, and improving overall usability to ensure a seamless and intuitive experience for users.
- **Partnerships:** We have gained AWS credits for our project and are seeking to partner with Microsoft for Startups. This collaboration will enable us to integrate and optimize OpenAI's GPT-4 easily within Linguist.

8.3 Potential Business Strategies

We are considering three potential business strategies, each with its own set of advantages and challenges. We may implement one or a combination of them: In-app purchases, a subscription model and in-app advertisements.

In-app purchases would allow users to pay for additional functionalities within the app. This approach offers the advantage of generating direct income from users who are willing to invest in enhancing their language learning experience. Purchasable items could include premium features or additional content that keeps users engaged and potentially increases learning efficiency. However, a downside of this model is that non-paying users may feel disadvantaged if important features are locked behind purchases. Additionally, managing in-app purchases requires implementing additional security measures and providing customer support, which adds to the operational complexity.

A subscription-based approach with a free trial period would allow users to access the main features of Linguist for free, with the option to subscribe for access to premium features. The recurring revenue stream from subscriptions provides financial stability and growth potential, and it's easier to upsell premium features to an existing customer base. However, there's a risk that lower-budget users may be unable to access essential features, and free trial offers may be abused, impacting revenue.

Alternatively, we could explore monetization through advertisements. This model would offer all features of Linguist to users free of charge, with revenue generated from third-party advertisements displayed within the app. Free features may attract more users, increasing the likelihood of viral growth and better brand recognition. However, advertisements may negatively impact the credibility and user experience of the app. Furthermore, there's a dependency on third-party advertisers for revenue, and the use of extensive cookies may raise privacy concerns among users.

Each of these potential business models presents unique opportunities and challenges, and the decision will ultimately depend on factors such as target audience preferences, revenue goals, and user experience priorities.

8.4 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 [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/ [Accessed: Mar 12, 2024].