Bilkent University

Department of Computer Engineering

Senior Design Project

T2533

StreamLang

Analysis and Requirement Report

22203632 - Uygar Bilgin - uygarblgn13@outlook.com
22202995 - Mehmet Emin Avşar - emin.avsar@ug.bilkent.edu.tr
22202913 - Göktuğ Ozan Demirtaş - goktugozandemirtas@gmail.com
22203805 - Ruşen Ali Yılmaz - rusenaliyilmaz@gmail.com
22203239 - Can Tücer - can.tucer@ug.bilkent.edu.tr

Uğur Doğrusöz
Mert Bıçakçı
İlker Burak Kurt

15.12.2025

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.

1. Introduction

This document introduces an overview of the Analysis and Requirements of the StreamLang system. In Section 2, the proposed system, the problem that StreamLang aims to address and the scope of the application is covered. The functional requirements and the non-functional requirements give a precise view of StreamLang's purpose. Within Section 2.5, System Models, each module is discussed separately and related UML modules are given to make concrete the design ideas overarching StreamLang's system architecture, alongside its coarse implementation details. In Section 3, Other Analysis Elements, the report discusses other important aspects like Risks and Alternatives and the ethical choices we will need to consider as a team. In addition, a vision for teamwork during the following weeks is laid out. Finally, the references are given in Section 4.

2. Proposed System

2.1 Overview

(Disclaimer: 2.1 Overview, 2.2 Functional Requirements and 2.3 Non-Functional Requirements are taken from the Project Specification Document with minor changes - since they overlap with that document and there have been no significant changes within the short period of time since the creation of that document)

Language learners require listening content to complement their studies; however, it is a pain to find content that is suitable for one's level. If you find some listening content that is pushing you to grow in terms of understanding complex grammar structures, it might be too easy in terms of vocabulary, or vice versa. Our solution allows the language learner to select precise grammar structures, specific vocabulary, content type, speaking speed and voice which are then used to create a listening stream uniquely tailored for the user's growth. Selected vocabulary and grammar structures are sprinkled into the listening stream using the scientifically proven *Spaced Repetition System* to optimize memory recall. Initial listening stream configuration templates are available as well as a Dynamic Mode which updates the vocabulary and grammar level automatically in time, enabling a passive learning experience.

2.1.1 Spaced Repetition Scheduler

For long-term learning, it is optimal to make sure that any learned information is repeated back to the learner in intervals to make sure that old information is not forgotten over time. A spaced repetition scheduler (SRS) is an algorithm that tries to calculate this interval. For our project, we will use the free spaces repetition scheduler algorithm (FSRS), which learns the user's memory patterns and calculates the interval for each piece of information the user has learned [1]. Specifically, this algorithm uses a model representing human memory with three variables:

- Retrievability (R): Probability that the learner can recall specific information at a specific time.
- Stability (S): Time required for R to decrease from 100% to 90% in days.
- Difficulty: Complexity of a given information.

The details of how these variables are calculated changes depending on the version of FSRS used, but generally the algorithm depends on calculating S using the other two variables and machine learning [1].

2.2 Functional Requirements

The application's main focus is creating structured listening streams for language learners that will help them attain fluency faster compared to listening to content not personally curated for them. The application will have a strong grammar-centric architecture where not only the vocabulary, but also the grammar structures that the user is exposed to will be tracked and then reintroduced at optimal times for memory retention according to SRS (Spaced Repetition System) principles, a method that has been shown to be effective in retaining information [2].

Dynamic Mode -for users who are willing to use it- will remove the need for the users to set up configuration templates themselves. In this mode, the user will first go through a brief assessment. According to the results of this assessment, a listening stream configuration will be set up, and then the configuration will automatically be updated in time as the user grows in their language abilities. User's growth in language skills will be assessed with intermittent quizzes administered through audio; if the user replies correctly to a question regarding the past tense, for example, their predicted skill in that grammar structure will increase and they will start to hear less of past tense and more of other, not yet learned grammar structures in their listening stream.

The specific scenarios that the user must be able to perform are listed as follows:

- Select the level of the vocabulary
- Select the type of content they are going to listen to
    - Article, story, individual sentences, translation of uploaded document, or some general text in the vein of the uploaded document
- Select whether they want to hear the translations
    - They can select to hear the translations after each sentence (only option for "individual sentences" type of content), or after the entire content
- Select the grammar structures they want to study.
    - Ex.: Past tense, present tense, perfect tense etc.
    - Select whether they want "Drill" mode or "Roaming" mode. In Drill mode, every sentence will cycle through the selected grammar structures. In "Roaming" mode these grammar structures will be interspersed throughout the listening.
- Select which voice they want to hear.
- Select the speed of the voice.
- Select whether they want the app to ask questions.
    - Select the frequency of the questions –after every sentence, after X sentences.
    - Select the type of questions they want the app to ask: comprehension questions, vocabulary questions (what is the meaning of X), "Construct this sentence in your target language" questions.
- Select which Anki account to integrate (optional)

- Choosing vocabulary from Anki deck overrides the “level of vocabulary” selection.
- Select whether they want card status in their Anki deck to be updated when:
  - (A) just by the listening stream going on for some time
  - (B) after answering “what is the meaning of this word” questions from the app
- Select whether new cards should be created in their Anki deck for words that they frequently come across during the listening stream.

Listening stream configuration templates catering to users with different skill sets and levels in the language will be available.

2.3 Nonfunctional Requirements

2.3.1 Usability

A quality application should be easy to use. Session tokens shall be cached on the user’s device to prevent launching the app from being tedious. Any audio to be streamed to the device shall download faster than its being played and all downloaded audio shall be fully cached for one playback session to facilitate scrubbing.

2.3.2 Reliability

Assuming continuous third-party service availability, the servers shall have at least 95% uptime and work without disturbance when not in maintenance. All data shall be constantly updated to ensure no data loss in case of a failure. The server data shall always be in a valid state. In case of malformed data, any requested operations using such data shall be denied. To avoid loss/corruption of data, any modifications to the same entry on server data shall be sequenced and not run in parallel. In case of data corruption or an invalid operation, the server shall be able to roll back the database to a valid state.

2.3.3 Performance

A smooth user experience is key for a quality application. The mobile client shall never show stutters, and any performance-heavy operation shall be completed in the background with a loading indicator. The server should also be Constantly/routinely operating modules shall not meaningfully impact the operating speed of other modules. Some user requests may require content to be generated using AI models. Any request that does not generate content shall respond to user requests under one second (excluding client-side network delays). Any content generating request shall respond to the user under one second after AI’s work is done.

2.3.4 Maintainability

StreamLang’s server’s modular architecture design allows for easy maintenance. When modules in need of maintenance can function independently of each other, application functionality that is isolated from any affected modules may keep running unaffected, while

any fixes/rollbacks are carried out on problematic modules. In the case of horizontally scalable modules, when a module instance has an outage/must be taken down, another instance may take its place to serve users until the instance is back up. Server operations shall be logged to make bug-fixing easier. Client maintenance is a non-issue since the entire application is already built and on the device of a user. Any updates shall be delivered through the device's application store.

2.3.5 Scalability

The client is a mobile application programmed in Flutter, so there will be no extra servers for the client application, meaning it can effectively be scaled infinitely. The backend will be separated into multiple modules to facilitate horizontal scaling. Any module that will be directly facing the user and is running strictly locally on our own servers shall have the ability to run in multiple instances and can be scaled infinitely. Modules using third-party services shall be scaled horizontally so long as the service provides such an option. With this architecture we should be able to adequately scale our application as user numbers change.

2.4 Pseudo Requirements

After discussions, we have settled on a few things regarding how we will implement and deploy the client application and backend.

The client application shall be programmed in the Dart language. The application shall be developed using the Flutter SDK. The application shall be supported by phones running the iOS and Android operating systems. The minimum required iOS and Android versions are to be decided later.

The backend shall be programmed in the Python language. The server modules shall be compatible with computers using both ARM and x86-based processors. The modules shall also run both on the Windows and Linux operating systems. PostgreSQL shall be used as the database solution to store anything that will not be using object storage. For generating speech audio, the backend shall run the Chatterbox model. Generated audio files shall be stored using an object storage solution that will be decided later.

2.5 System Models

2.5.1 Scenarios and Use Case Model

There is one type of user persona that StreamLang is aimed at: Language Learner. However, there may be several types of language learners: beginners, intermediate, and advanced learners.

Beginner and intermediate language learners may come to have different expectations from the application. Therefore, their different expectations are enumerated in the various options we supply the user in configuring the listening stream themselves or choosing a template. There is only one *ex rigore* use case of the application: creating and listening to a stream personalized for the user. There are different ways the Language Learner may achieve this use case, laid out below in use cases written according to the UML specification. They are then presented visually in a Use Case Diagram.

StreamLang aims to do one thing, creating personalized language learning audio streams, very well. The necessary complications reflect more on the backend structure than the scenarios.

The question might be asked; why is there only a single actor, the Language Learner. Are there no System Administrators? No, there are no System Administrators, since the program is for every individual's personal use. There is no team concept or team management dynamics. Any content is either custom created by the Language Learner or added to the application by the developers. An extension of the application in the future specialized for education purposes, involving a teacher and this teacher assigning certain listening streams to their students may come into being.

The term "clicks on" is used below to capture a general sense of interaction. It can be interpreted as "taps on" as the application runs on a Mobile Client.

Use Case: Create Account

Actor: Language Learner

Preconditions: Language Learner has installed the application to their mobile phone.

Main Flow:

1. Language Learner opens the mobile application.
2. Language Learner clicks "Create Account".
3. Language Learner enters their name, email, and payment details.
4. Language Learner confirms their email by clicking on the one-time magic link sent to their email.
5. Mobile Client directs Language Learner to the Onboarding page (see UI Mockups).
6. Language Learner selects their native language and target language.
7. Mobile Client directs Language Learner to the Templates page (see UI Mockups).

Postconditions: Language Learner has created and logged into their account.

Use Case: Create and Listen to a Audio Stream for Language Learning Based on a Template

Actor: Language Learner

Preconditions:

- Language Learner has logged into their account.
- Language Learner is on the Templates page.

Main Flow:

1. Language Learner searches for their desired Template by scrolling AND/OR plain text search AND/OR filtering.
2. Language Learner clicks on the Template they want to listen to.
3. Mobile Client directs to the Streaming page.
4. Language Learner listens to the Audio Stream.
5. Language Learner decides they are done listening and closes the stream.

Postconditions:

- Language Learner's State (where they left off listening) is recorded to the Database.
- Language Learner's Exposure to the vocabulary and the grammar that are contained in the portion of the stream that they have listened to is recorded to the Database.

Use Case: Listen to an Audio Stream for Language Learning Based on a Template That Has Been Listened to Before

Actor: Language Learner

Preconditions:

- Language Learner has logged into their account.
- Language Learner clicked on a Template that they have listened to before

Main Flow:

1. Mobile Client loads Language Learner's State in that Template.
2. Mobile Client directs to the Streaming page.
3. Language Learner starts listening to the Audio Stream initialized on a point based on their previously loaded State.
4. Language Learner decides they are done and closes the stream.

Postconditions:

- Language Learner's State (where they left off listening) is recorded to the system.
- Language Learner's Exposure to the vocabulary and the grammar that are contained in the portion of the stream that they have listened to is recorded to the Database.

Use Case: Create and Listen to a Personalized Audio Stream for Language Learning

Actor: Language Learner

Preconditions: Language Learner has logged into their account.

Main Flow:

1. Language Learner clicks “New Session” on the sidebar menu.
2. Mobile Client directs Language Learner to the Stream Configuration page.
3. On the Stream Configuration Page, Language Learner chooses to configure or ignore all the configuration options enumerated under Section 3.2 (Functional Requirements) of this report.
4. Language Learner clicks on “Start Listening”.
5. Mobile Client directs to the Streaming Page.
6. Language Learner listens to the Audio Stream.
7. Language Learner decides they are done and closes the Stream.

Postconditions:

- The new listening stream configuration is recorded in the Database. It will be listed on the sidebar of the application for quick future access.
- Language Learner’s State (where they left off listening) is recorded to the system.
- Language Learner’s Exposure to the vocabulary and the grammar that are contained in the portion of the stream that they have listened to is recorded to the Database.

Use Case: Listen to a Previously Configured Stream for Language Learning

Preconditions:

- Language Learner has logged in to their account.
- Language Learner has previously created a Listening Stream configuration.

Main Flow:

1. Language Learner opens the sidebar where Listening Stream Configurations are listed.
2. Language Learner clicks on the Template that they want to listen to.
3. Mobile Client directs to the Streaming Page.
4. Mobile Client initializes the Audio Stream according to the previously recorded State of the Language Learner regarding that specific configuration.
5. Language Learner listens to the Audio Stream.
6. Language Learner decides they are done and closes the Stream.

Postconditions:

- Language Learner's State (where they left off listening) is recorded to the system.
- Language Learner's Exposure to the vocabulary and the grammar that are contained in the portion of the stream that they have listened to is recorded to the Database.

Use Case: Listen to a Stream in Dynamic Mode for the First Time

Preconditions: Language Learner is logged in to their account.

Main Flow:

1. Language Learner clicks on “Dynamic Mode” from the menu.
2. Mobile Client assesses the level of the Language Learner through a short quiz.
3. Mobile Client populates a listening stream configuration based on the results of the quiz.
4. Mobile Client directs Language Learner to the Streaming page.
5. Language Learner starts listening to the Audio Stream.
6. Language Learner decides they are done and closes the Stream.

Postconditions:

- Language Learner’s State (where they left off listening) is recorded to the system.
- Language Learner’s Exposure to the vocabulary and the grammar that are contained in the portion of the stream that they have listened to is recorded to the Database.
- According to Exposure to the grammar points and vocabulary included in the part of the Listening Stream that Language Learner has listened to, the Dynamic Listening Stream Configuration is updated automatically.

Use Case: Listen to a Stream in Dynamic Mode

Preconditions: Language Learner is logged in

Main Flow:

1. Language Learner clicks on “Dynamic Mode” from the menu.
2. Mobile Client initiates the Audio Stream according to the previously recorded State of the Language Learner regarding Dynamic Mode.
3. Language Learner starts listening to the Audio Stream.
4. Language Learner decides they are done and closes the Stream.

Postconditions:

- Language Learner’s State (where they left off listening) is recorded to the system.
- Language Learner’s Exposure to the vocabulary and the grammar that are contained in the portion of the stream that they have listened to is recorded to the Database.
- According to Exposure to the grammar points and vocabulary included in the part of the Listening Stream that Language Learner has listened to, the Dynamic Listening Stream Configuration is updated automatically.

Use Case: Login

Preconditions: Language Learner has previously created an account.

Main Flow:

1. Language Learner opens the application.
2. Language Learner inputs their email and password.
3. Language Learner clicks or does not click on “Remember Me”.
4. Language Learner clicks on “Login”.
5. Mobile Client directs Language Learner to the Templates page.

Postconditions: Language Learner is logged in.

A pedestrian recounting of all the commonplace scenarios regarding login is avoided for the sake of brevity. There is rate limiting and password reset using email, handled according to industry standard ways.

Use Case: Delete Account

Preconditions: Language Learner is logged in to their account.

Main Flow:

1. Language Learner opens the sidebar.
2. Language Learner clicks on Settings.
3. Language Learner clicks on “Delete Account”.
4. Mobile Client deletes the Account (through interacting with the backend).

Postconditions: Language Learner’s account is deleted.

Use Case: Delete Data

Preconditions: Language Learner is logged into their account.

Main Flow:

1. Language Learner opens the sidebar.
2. Language Learner clicks on Settings.
3. Language Learner clicks on “Delete Data”.
4. Language Learner selects the Data they want to delete.
5. Language Learner clicks on Delete.
6. Mobile Client deletes Language Learner’s data (through interactions with the backend).

Postconditions: Language Learner’s data is deleted.

Use Case: Delete an Existing Listening Stream Configuration

Preconditions:

- Language Learner is logged into their account.
- Language Learner is on the Listening Stream Configurations Page.

Main Flow:

1. Language Learner slides the Configuration in card view to the left.
2. A red button including a minus appears.
3. Language Learner clicks on the red button.
4. Mobile Client deletes the Configuration (through interacting with the backend).

Postconditions: The Listening Stream Configuration is deleted.

Use Case: Share a Listening Stream Configuration

Preconditions:

- Language Learner is logged into their account.
- Language Learner has previously created a Listening Stream Configuration.

Main Flow:

1. Language Learner navigates to the Listening Stream Configurations page.
2. Language Learner clicks on the three dot on the card view of a Configuration.
3. Language Learner clicks on “Share”.
4. The link of the listening stream configuration is copied into Language Learner’s clipboard.
5. Language Learner shares the link.

Postconditions: The shared Listening Stream Configuration is made Public.

These use cases can be represented in a Use Case Diagram as follows:

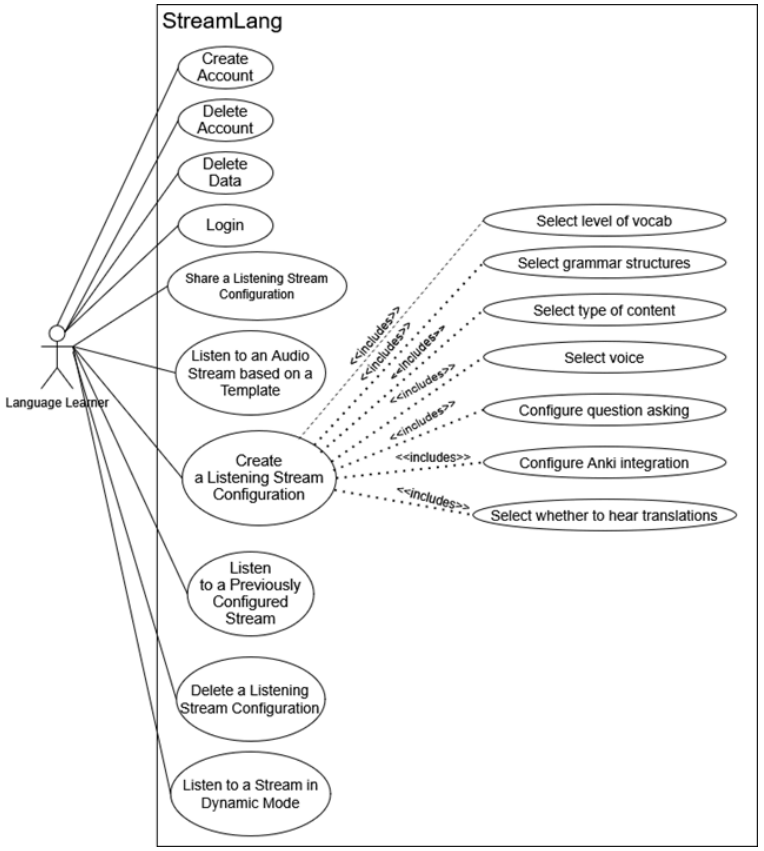

Fig. 1. StreamLang Use Case Diagram

2.5.2 Object and Class Models

2.5.2.1 Database Design

Database schemas should be designed in a way as to form a solid basis for the content scheduling function of StreamLang. In Fig. 2, we see the E/R diagram of the StreamLang database.

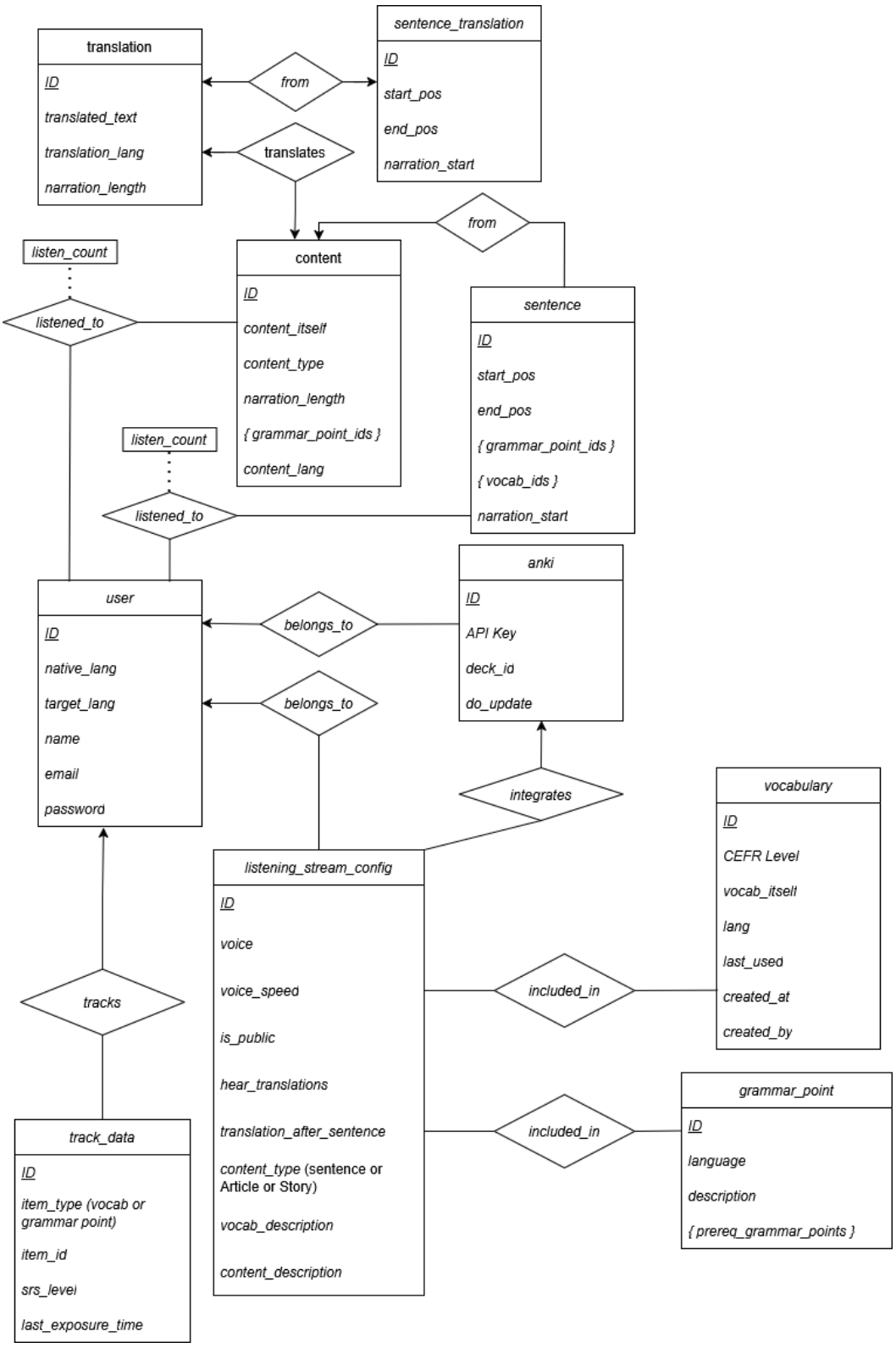

Fig. 2. StreamLang Entity Relationship Diagram

The explanations of each entity and relation set is as follows. In order for the explanations to be comprehensible, the term *SRS Level* should be defined. In the context of StreamLang, *SRS Level* has roughly the same meaning as *proficiency level*. SRS Level dictates how frequently the user is going to be exposed to a certain grammar point or vocabulary.

During the explanations, several details are given regarding how some tables are populated or managed. These details are expounded upon in their respective sections and are mentioned here briefly only because they are relevant.

user: This entity represents the user of the application, the Language Learner.

listening_stream_config: This entity is how we store listening stream configurations in the database.

vocabulary: This entity is how specific vocabulary are stored in the database. New vocabulary is inserted into the database through two ways. Either the user, while defining a configuration, explicitly lists a vocabulary to be included (and tracked) in the listening stream, or describes the type of vocabulary and the System creates those vocabulary and inserts them into the database. *vocabulary* rows that have not been used for a long time are periodically deleted. There is a limit to how many vocabulary a user can add to the system within a given period of time.

grammar_point: This entity represents a grammar point. The “prereq_grammar_points” is a multivalued field that is useful for Dynamic Mode. In Dynamic Mode, when the user achieves a good SRS level on a certain grammar point, the next grammar point that will be introduced into the listening stream will be chosen from the set of grammar points whose prerequisites have now been satisfied.

track_data: Arguably the most important entity. Tracks what SRS level a certain user is on regarding a specific grammar point or vocabulary.

anki: An entity for Anki integration data, with *integrates* relation with the user.

content: This entity represents the scraped content from the Content Scraping module. An entire journal article, or a story is stored here.

sentence: This entity is connected with *from* relation to *content*; it represents the individual sentences of the scraped content. A regular batch job in the server, the Content Categorization Module, separates the scraped content into individual sentences and labels them according to their grammar points and vocabulary.

listened_to: This is a relation between *user* and *content* (and individual *sentence*s of *content*). The specific *listen_count* is being tracked in this relation as well. The purpose of this relation is that: when the user is about to be served content including a certain grammar point (or vocabulary), they will ideally be served content that they have never listened to before or, if they listened to everything, the content they listened to the least.

belongs_to: This is a relation set simply stating that the user owns their Anki account and the listening stream configurations that they themselves have created.

translation and *sentence_translation*: These are entity sets mirroring the *content* and *sentence* entity sets and contain the translations of these.

2.5.2.2 Class Design

The class diagram allows us to see interactions between different class instances for StreamLang. In Fig. 3, we can see the class diagram for the project.

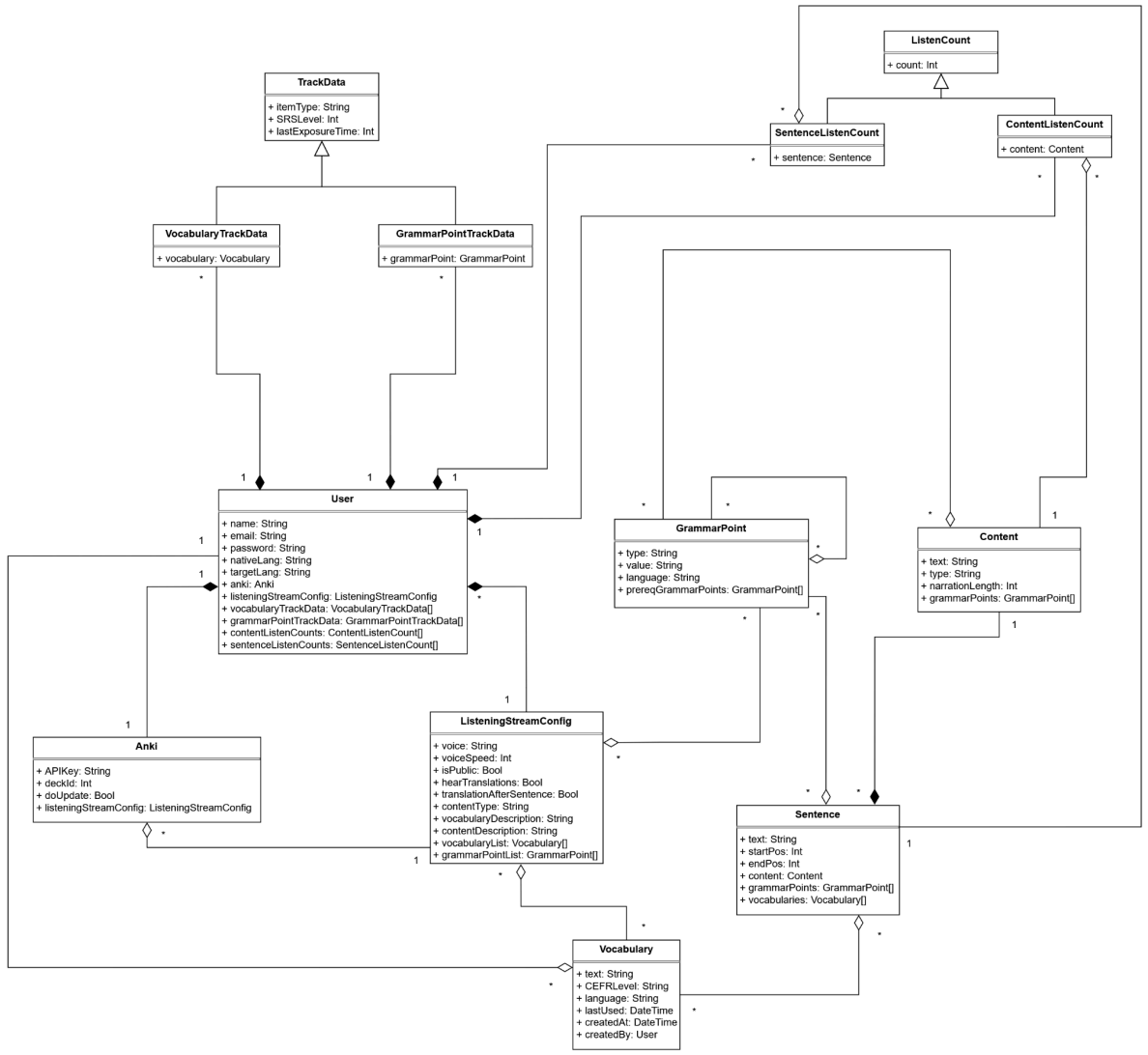

Fig. 3. Class Diagram for Content Categorization Module.

Here, explanations for each class are given below.

TrackData: Parent class for **VocabularyTrackData** and **GrammarPointTrackData**. Contains tracking information for last exposure time to a vocabulary or grammar point, and what SRS level the user is currently at.

VocabularyTrackData: Inherits from **TrackData**. Has an extra attribute of the vocabulary being tracked.

GrammarPointTrackData: Inherits from **TrackData**. Has an extra attribute of the grammar point being tracked.

Anki: Tracks Anki integration data. Has attributes for API key, Anki deck ID, current listening stream config, and whether the deck should be updated.

ListenCount: Tracks listen counts for sentences and content.

SentenceListenCount: Inherits **ListenCount**. Contains the sentence being tracked.

ContentListenCount: Inherits **ListenCount**. Contains the content being tracked.

Content: Contains information about a specific content. Has attributes for the content text, content type (story or article), content length, and grammar points found in the content (those that pass the 5% threshold).

GrammarPoint: Contains information about a specific grammar point. Has attributes for the grammar point type (case, tense, etc.), value for that type (future tense, accusative case, etc.), language for the grammar point, and prerequisites for that grammar point, which are the grammar points with less complexity than that grammar point, and such, need to be learned beforehand.

Sentence: Contains information about a specific sentence. Has attributes for the text of the sentence, start and end positions in the content the sentence is in, the content sentence is from, grammar points found in the sentence, and vocabularies found in the sentence.

Vocabulary: Contains information about a specific vocabulary. Has attributes for the vocabulary text, CEFR level of the vocabulary, language of the vocabulary, the time the user last used the vocabulary, creation time of the vocabulary, and the user who created the vocabulary.

ListeningStreamConfig: Contains information about a stream config. Has attributes for voice of the stream, speed of the voice, whether the config is public or not, whether the user wants to hear the translations, whether there should be a translation after every sentence, content type of the stream, description of the vocabularies wanted in the stream, description of the content wanted for the stream, list of vocabularies wanted for the stream, and a list of grammar points wanted in the stream.

User: Contains information of a user. Has attributes for user login data, native and target language of the user, anki integration, current listening stream config, track data for vocabulary and grammar points, and listen counts for content and sentences.

2.5.2.2.1 Frontend Class Diagram

The frontend does not store much information. The major classes are the User data, session configuration, language metadata, and stream state.

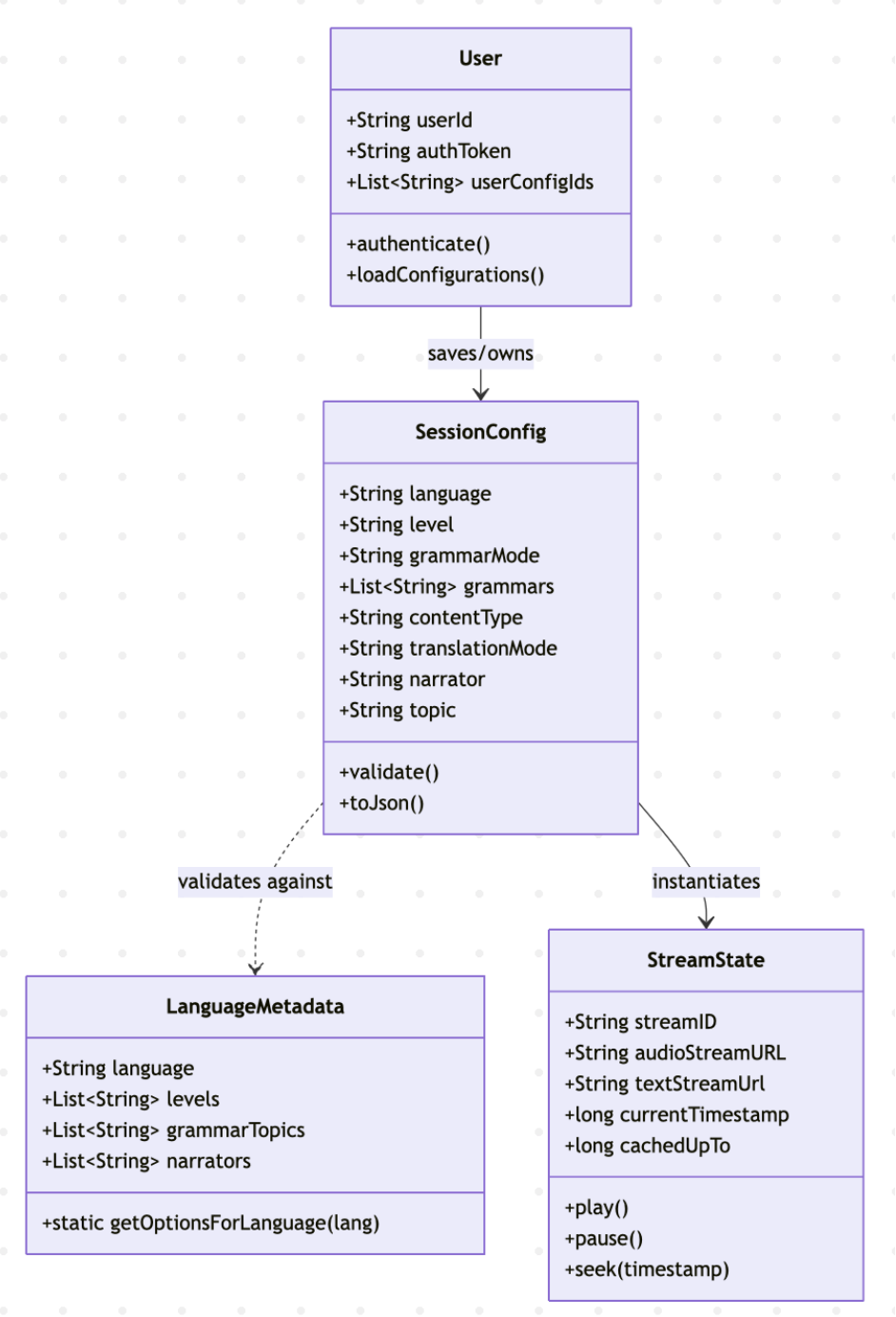

Fig. 4. StreamLang Frontend Class Diagram

2.5.3 Dynamic Models

2.5.3.1 Content Scheduling of the General Backend

The LLM orchestration module and the scheduling function of the backend jointly constitute the heart of the application. By scheduling function, we mean the decision mechanism of the backend regarding when a certain grammar point or vocabulary will be introduced into the listening stream, and through which content or sentence that point will be introduced. In this subsection, through activity diagrams, we inquire into how content is served to the user and when the requested content is too specific, we fallback into an orchestrated LLM mechanism to generate the content the user desires.

In the diagrams below, when the phrase “Content is served to the User” or “Sentence is served to the User” is used, the precise meaning is: “Streaming Module is instructed to serve Content / Sentence to the User”. The Streaming Module has its own mechanism of doing this serving (based on the availability of narration in object storage), which is explained under its own section.

The scheduling of content is going to change depending on whether the user chose to listen to entire contents (an entire article, story etc. with discernible beginnings and ends) or individual sentences. First, let’s look at the activity diagram when the user wants to listen to entire contents, not individual sentences.

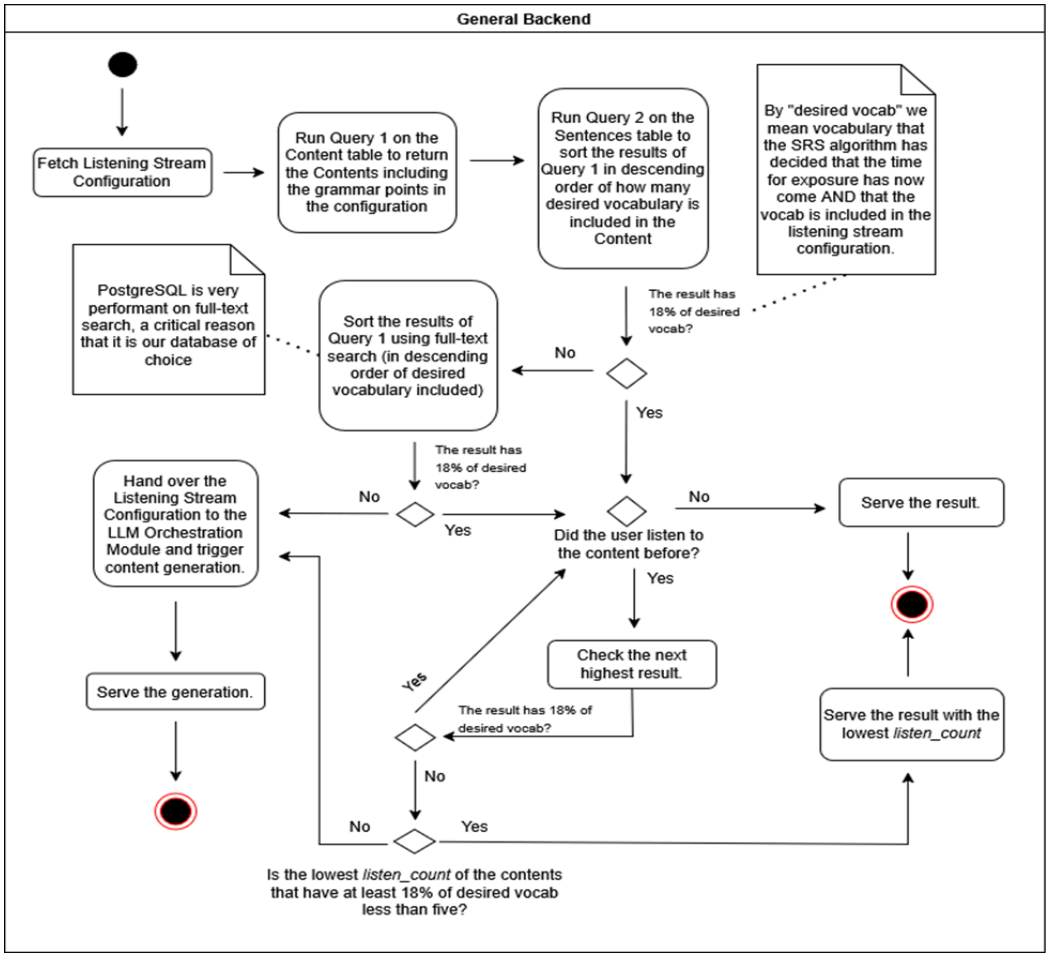

Fig. 5. StreamLang General Backend Activity Diagram

Since finding a pre-fetched content that matches user requirements perfectly is difficult, the system considers a threshold of 18% (whose value we as developers may decide to change in the future depending on empirical results of user experience), and serves the user the content that has at least 18% of the desired vocabulary. Desired vocabulary is defined as the vocabulary that the SRS algorithm has decided time for exposure has come. After streaming this content, the vocabulary that was included in this content will not be marked as “time for exposure has come”, therefore, the newly selected content will have to include 18% of the other, not already listened to vocabulary that is included in the listening stream configuration.

When we consider serving the user individual sentences instead of entire contents with a beginning, middle, and an end, the activity diagram in Fig. 6 below represents how relatively simple the job becomes.

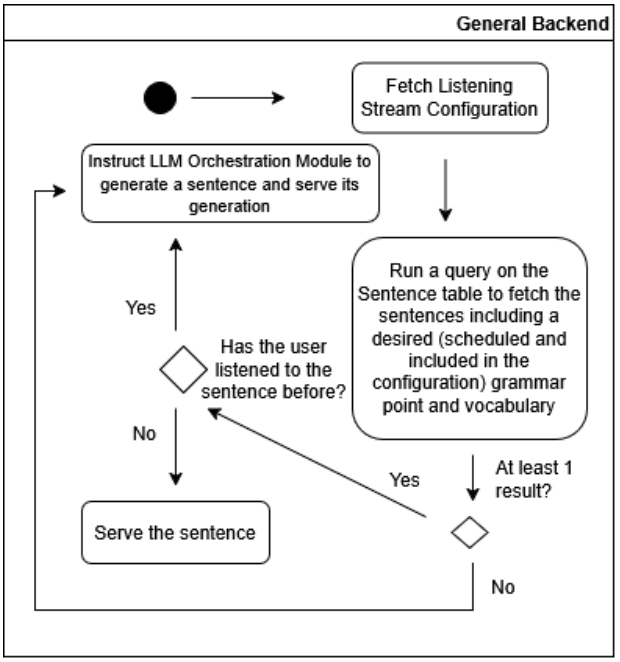

Fig. 6. StreamLang Streaming Activity Diagram

The activity diagram represents the activity of the general backend; it searches the Sentence table to fetch a sentence including a desired (scheduled by the SRS algorithm and included in the listening stream configuration) grammar point and vocabulary. If there are no results or there is a result but it has been listened to by the user before, we generate the sentence using the LLM orchestration module and serve the result. But the question remains: how does the LLM Orchestration module handle these generations? That is the topic of the next subsection.

2.5.3.2 LLM Orchestration Module

In some scenarios, the LLM Orchestration module will be served a listening stream configuration and be expected to produce content fitting its criteria. The general data structure for a listening stream configuration can be seen from the ER diagram above, at Fig. 2.

The fields relevant to the LLM Orchestration module are

- Whether the user will hear the translations of the sentences or not
- Whether the user will hear the translations after each sentence, or after the entire content is done (more metadata is needed in the former case)
- The content description and type desired
- The type of vocabulary described by the user as desired to appear in the content
- The specific vocabulary given by the user that should appear in the content.
- The specific grammar points given by the user that should appear in the content

Thus, we have six key information at hand: (1) translation wanted or not, (2) translation after sentence or entire content, (3) content description, (4) content type (which can be Article, Story, or Individual Sentences), (5) vocabulary description, (6) grammar points.

It should be noted that the vocabulary and the grammar points will be handed over to the LLM Orchestration Module by the General Backend Module. The General Backend Module will carefully assess and coordinate which vocabulary or grammar points are requested in coordination with the SRS algorithm; the scheduling will be handled there.

Given the six information points, the LLM Orchestration Module will just create one continuous text. There will be two LLM agents. One will be the Creator. The other will be the Reviewer. The Creator will create the content, which will then be channeled to the Reviewer. The Reviewer shall point out any deficiencies in the created Content. In case of a mistake, the Creator will be asked to recreate the entire content or the problematic parts of the previously created content with a modified prompt indicating the problems that should be avoided. During the review phase, the content will be separated into individual sentences, and each sentence will be evaluated by an LLM to see if it has any mistakes. This per-sentence check will be in addition to a review of the whole content.

The moment that a content is generated by the LLM Orchestration module, it will be inserted into the database with the related metadata; its narration is also commenced by a voice model and then saved to S3 Object Storage with a structured naming system for easy access afterwards. For the contents themselves, the audio files are saved with the name [*content_id*]. The translations are saved as [*translation_id*].

The Content Categorization Module will save metadata about individual sentences of the created content in the same way it does with scraped content.

2.5.3.3 Content Categorization Module

Content Categorization Module (CCM) allows us to categorize content based on grammar points. When a user requests a listening stream, the user is able to select grammar points or vocabularies to appear in the content. CCM serves this request by tokenizing sentences in the content, classifying their grammar points, and assigning grammar point categories to the content based on a 5% threshold. CCM also tokenizes sentences based on vocabularies found in the sentence, and if that vocabulary is requested by the user before, it assigns relations between sentences and vocabularies.

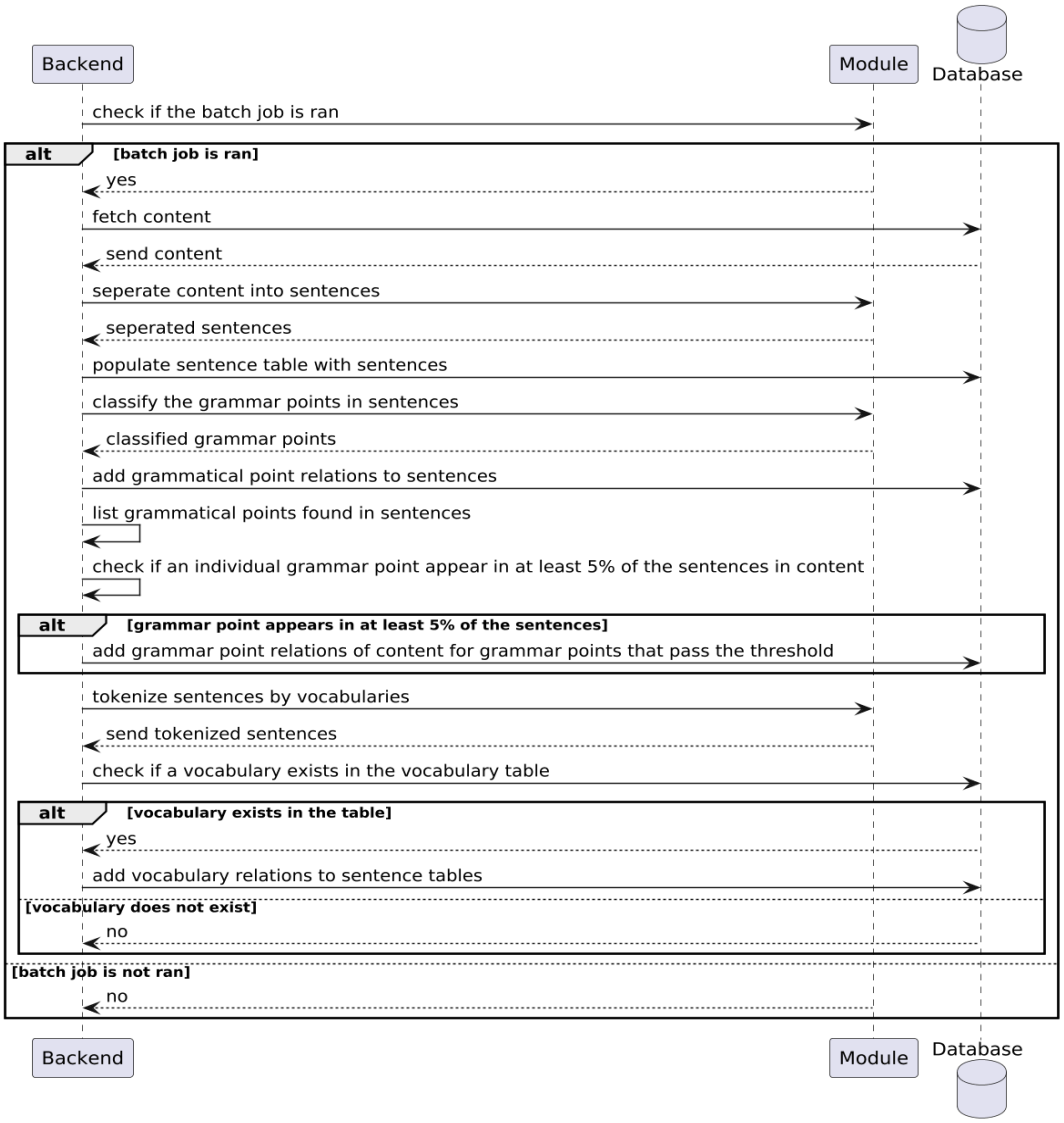

Fig. 7. Sequence Diagram for Content Categorization Module.

2.5.3.4 Streaming Module

The streaming module is the part of the server that will be responsible for sending the generated audio clips to the user in a stream on demand. It will not be responsible for the actual generation or storage of any audio. The audio will be streamed instead of being sent as a whole to give the user a faster and more seamless experience in case of long content and/or a slow internet connection.

When sending an audio file, we plan for the server to The generated audio will be taken from the S3 Object Storage system to be sent to the user. We chose an object storage system instead of dumping the data to our database because audio files are relatively large compared to the metadata we will be using our database for. Object storage solutions make storing large amounts of data relatively easy and cheap. S3 also has an API to give us the ability to stream this data to our servers instead of trying to download the entire file before we stream it to the user. This will both save on time and lower the bandwidth usage of the server per user, giving it room to serve more users.

2.5.3.5 Content Scraping Module

Content scraping module scrapes and stores content to be used either directly for audio stream generation or to be used for a baseline for on-demand original content generation via an LLM. It works by scraping raw text data from some given online source, processing it to remove any unwanted content, categorizing it according to the language of the text, and inserting this content into an appropriate database table. It is run and controlled by the general backend module. The content scraping job is periodically run to populate the contents table.

When run, the module iterates through a list of URLs to access those pages through HTTP and attempts to take the text content from the parsed HTML structure. The list of URLs is fetched from a database table, which is updated manually when needed. While storing scraped content, the module also saves a hashed key for the content, to be able to skip similar content without saving in the future. A sequence diagram showing the architecture of the content scraping module can be seen below.

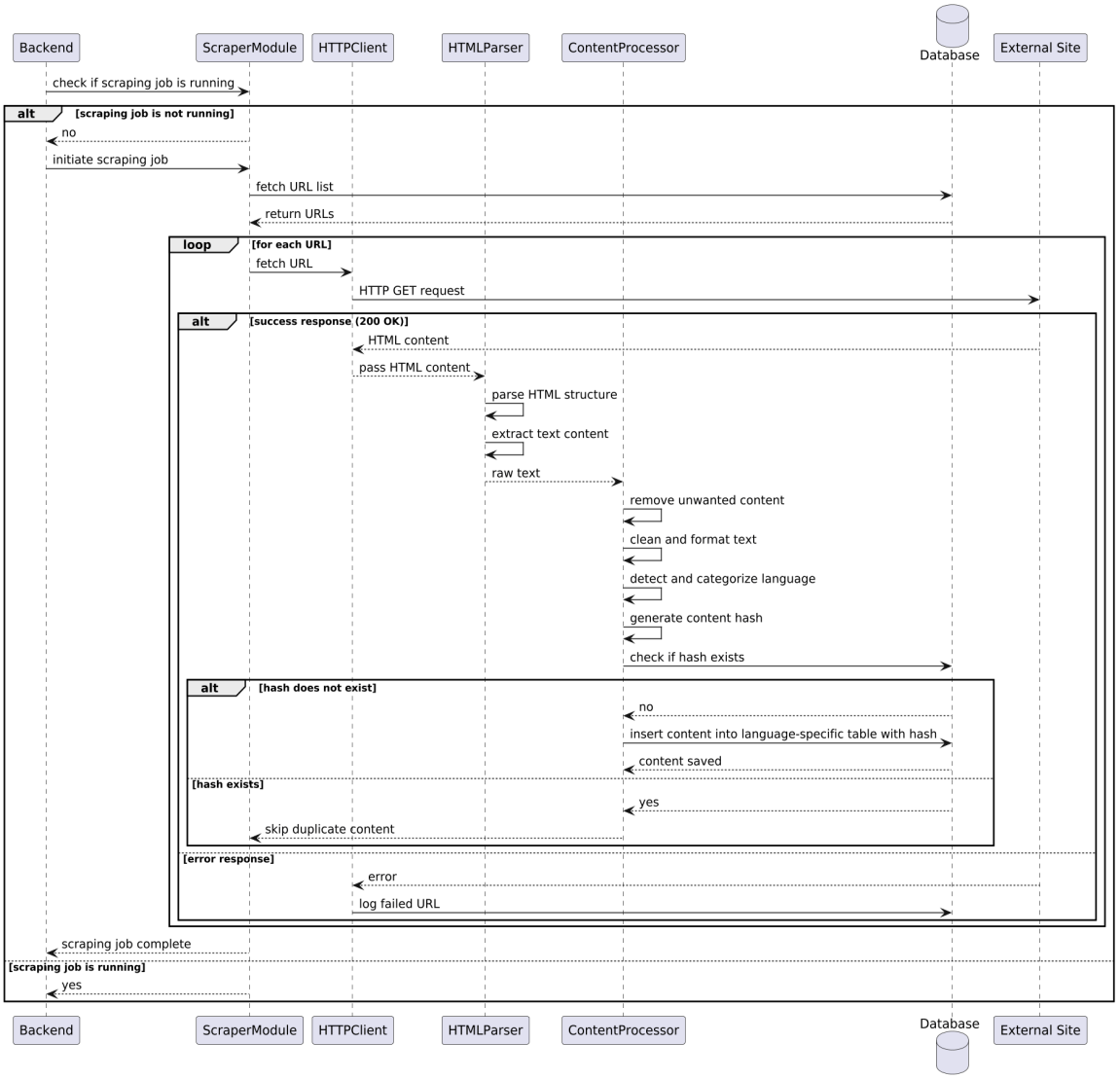

Fig. 8. Sequence diagram for content scraping module.

2.5.4 User Interface - Navigational Paths and Screen Mock-ups

User interface design is publicly available on Figma: *https://www.figma.com/design/XjpFndxeSeZ9YMrG7Eq1SU/Senior-Project-UI-Design--Public-V1-?node-id=0-1&t=GsU4WwzLosAyFYZ7-1*

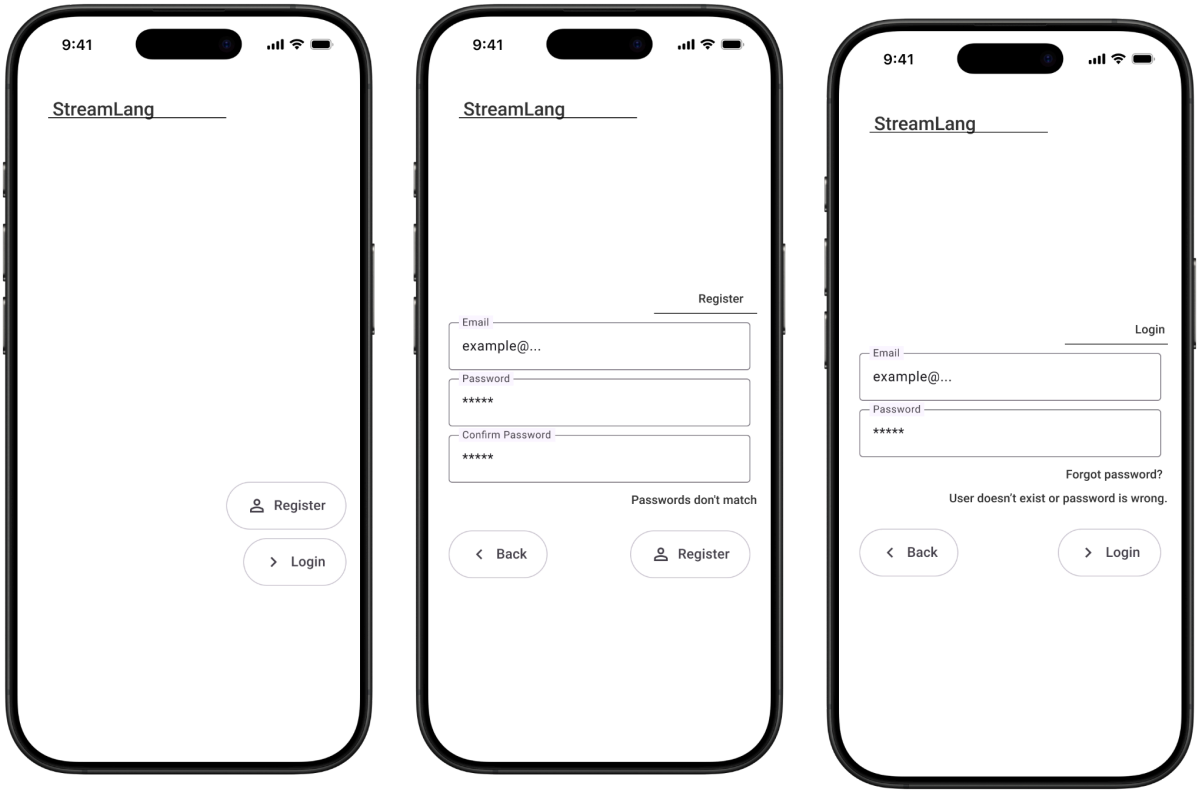

Fig. 9. Interface of authentication pages.

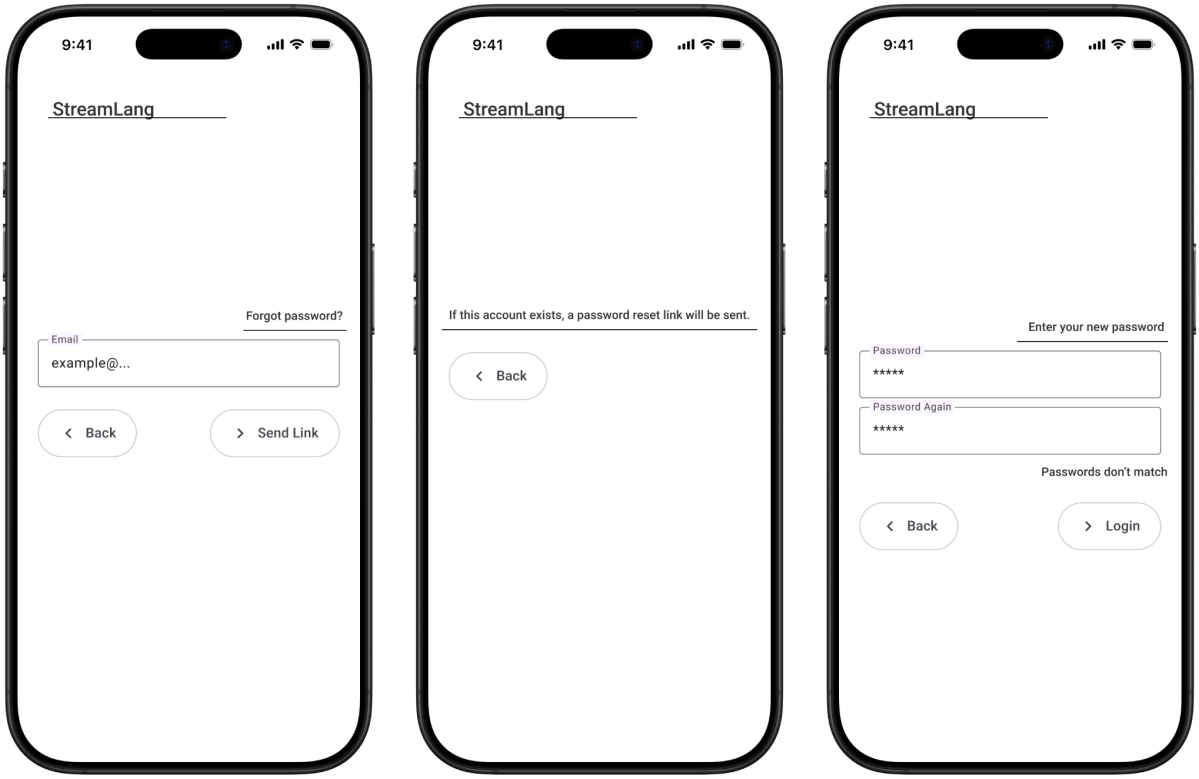

Fig. 10. Interface of password change pages.

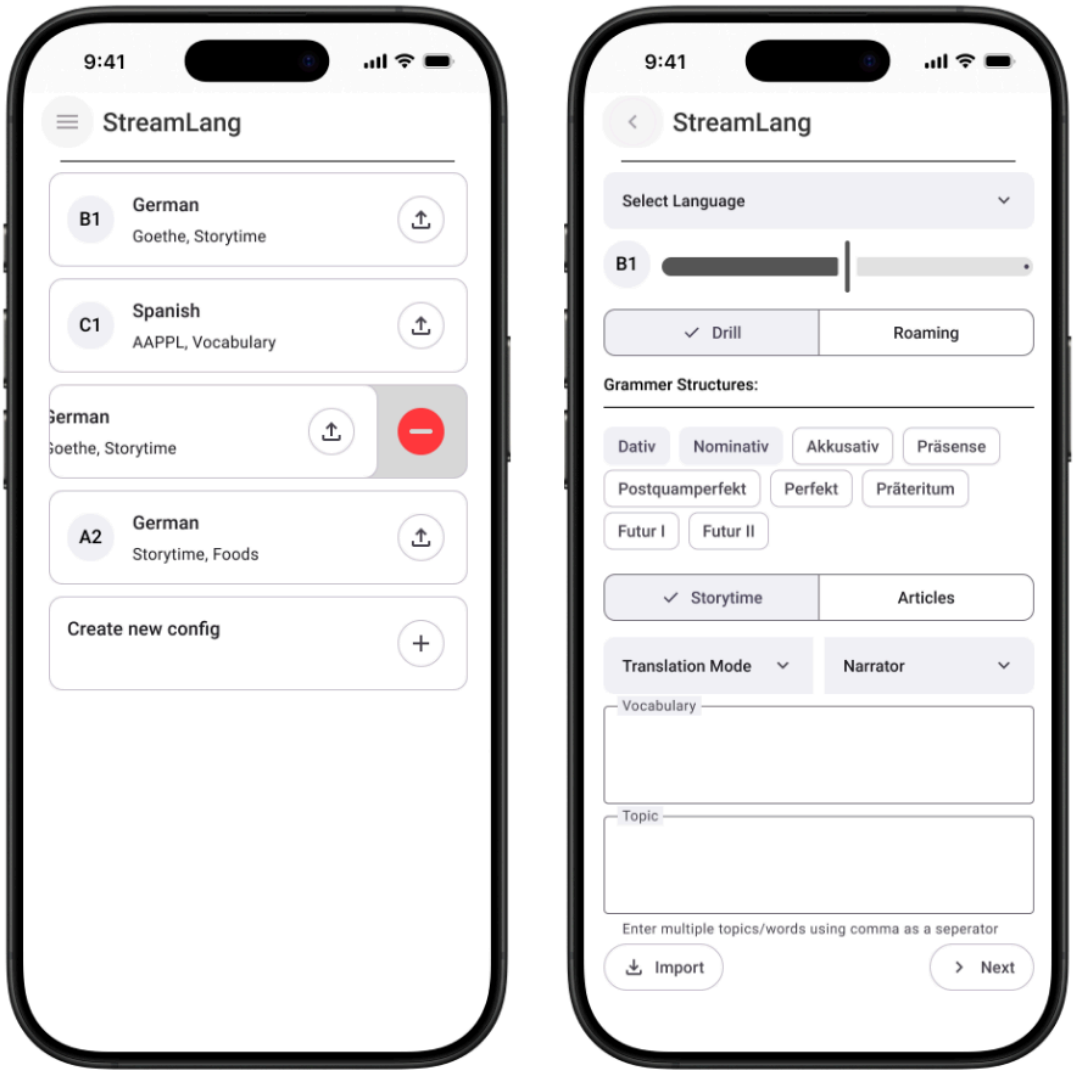

Fig. 11. The Interface for Template Configurations and Configuration Arrangement Page

Fig. 12. Interface of listening pages.

3. Other Analysis Elements

3.1. Consideration of Various Factors in Engineering Design

3.1.1 Constraints

With StreamLang we are aiming at the average language leaner. With the current market expectations we need a responsive cross platform app with good UX design. For this reason we chose Flutter for mobile app development. Flutter allows developers to write native code, such as Objective-C and Kotlin alongside Dart for better performance and a lower level access. Flutter also reduces the engineering load as it is a cross platform framework. We also need extensive search capabilities for the backend, while also staying efficient. But alongside efficiency, we also need compatibility with AI systems and ease of development. For these complex constraints we chose a combination of python, postgres, and redis for the backend. Python, although often not chosen for enterprise backend development, offers ease of development with industry standard packages available for AI integration. Although python can be more computationally expensive than alternatives, being able to run on almost any environment brings a significant advantage for deployment. Postgres offers both blob and standard storage, meaning it allows us to store both pre-generated audio files and metadata in a single database system. Redis on the other hand allows for in-memory caching for better response times, and reduced database load.

Sustainability is an important constraint for StreamLang since the project will utilize AI generation in order to match specific user requests, and AI generations are known to consume water in keeping data centers cool. In order to minimize AI generations as much as possible, the system will cache previously generated content and will not regenerate content for two very similar listening stream configurations. In addition to caching AI generations, the system will also first look into the already existing content library of copyright free texts before falling back to AI generation.

3.1.2 Standards

- [RFC 768] User Datagram Protocol : UDP streams will be used for transmission of the audio, as lossy transmission of audio has negligible impact on the resulting audio.
- [RFC 2326] Real Time Streaming Protocol : RTSP, and its respective built in controls, will be used to control the audio stream from the server.
- [RFC 7519] JSON Web Token : JWT’s will be used for authentication purposes.
- [FIPS 180-4] Secure Hashing Standard : SHA-2 will be used for storing credentials in the database.
- [RFC 6716] Definition of Opus Audio Codec : Opus codec will be used for the universal filetype for audio files.
- [RFC 2818] HTTP over TLS: Mobile operating systems often require https for network communications, and hence we will use that.
- Open API Spec : Open API specifications will be used for communication with LLM providers.
- For diagrams, we will use Unified Modeling Language version 2.5.1.

- [RFC 8259] The JavaScript Object Notation Data Interchange Format : The API communication will use JSON for notation of the data.
- [RFC 3629] UTF-8, a transformation format of ISO 10646 : As the product will support many languages, we will use a character encoding that supports as many international characters (and scripts) as possible.
- [ISO/IES 9075] Database Language SQL : We will be using SQL for communication and CRUD operations with the postgreSQL database.

3.2. Risks and Alternatives

With StreamLang, we aim to cater to a comprehensive percentage of language learners around the world. This requirement brings the constraint of creating a simple, usable, and effective application. As this is not a trivial task, this project brings with it many complications and risks that we will have to assess. In addition, as language learning is a large field, our application has several competitors and alternatives, even in the niche of listening stream generation. In this section, several risks we might encounter will be analyzed, and an analysis of alternatives that deliver a similar use case to our application will be made.

In our project, we aim to curate content for our users aimed to cater to their language levels and needs. One of the problems that might arise is failing to achieve that. We might not be able to find or generate content with specified grammar points, vocabulary, and type for the user. Another problem that might arise is a lack of accuracy for our LLM modules. The content we generate might not have the quality we want to serve to the users. Other than these, we might encounter problems with our text to speech module not working properly for some texts, or problems with scraping the content we want.

We have several competitors offering similar services to our application. One of them is Taalhammer [3], an AI powered spaced repetition app that offers a sentence based learning experience. Another competitor is Lenguia [4], an app that utilizes AI to generate personalized stories based on your language level and interests. It uses the comprehensible input approach for natural language learning, and utilizes methods such as spaced repetition. Lastly, we have Beelinguapp [5], offering parallel translated texts for two languages for their leveled content library, with synchronized native speaker audio. It also does a comprehension assessment at the end through quizzes and progress tracking for vocabulary mastery.

While competitors to our application exist, we have a few competitive advantages that make StreamLang stand out. These include grammar centric configuration, that is, giving our users the option to select specific grammar structures for content and showing content based on these grammar points. We also offer Anki [6] integration, updating Anki flashcards and generating new ones as the application is used. We also offer dynamically adapting our content, continuously assessing through questioning and automatically adjusting content difficulty. Finally, we offer configurable questions for comprehension, vocabulary and sentence construction for turning our application into an active learning experience. All these features help StreamLang to stand out from its competitors, giving it an edge over them.

In case of failure of the original vision of StreamLang due to inaccuracy of computer translations, the expensiveness of cloud GPU providers making the product too expensive and pushing out potential customers, or TTS working too slowly; our plan B is to put much more emphasis on preparing a vast audio library using automated processes on scraped data, and retrieving that instead of focusing on generation.

Table I
Risks.

Risk	Likelihood	Effect on The Project	B Plan Summary
Inaccurate LLM	Medium	Inaccurate translations and incoherent text	Using scraped content data
TTS Failures	Low	Inaccurate listening stream	Searching for alternative TTS modules, or using prerecorded audio libraries
Too Expensive Cloud Providers	Low	Inability to feasibly deliver generated content	Using scraped content data
Problems With Scraping Data	High	Inability to deliver human-made literary content	Using generated content data

3.3. Project Plan

Below is a list of our project goals, sorted by planned completion date.

TABLE II
List of project goals.

Goal Definition	Planned Completion Date
Finalizing requirements and decisions.	21.11.2025 (completed)
Preparing Project Specification Document.	21.11.2025 (completed)
Preparing Analysis and Requirements Report.	12.12.2025
Developing the first working prototype.	Middle of December
Preparing the demo presentation.	Late December
Releasing the first version of our system with all listed requirements.	Early February
Releasing the final version in the most polished state.	Early April
Preparing the High-Level Design Report.	Early May

To be able to achieve our goals in the planned times, we will divide our work into the following packages.

TABLE III
List of work packages.

WP#	Work Package Title	Leader	Members Involved
1	Finalization of project scope, features, and technical details.	Mehmet Emin Avşar	All members
2	Completion of UI and interaction designs.	Ruşen Ali Yılmaz	Göktuğ Ozan Demirtaş, Can Tücer
3	Development of the data scraping algorithm.	Can Tücer	Uygar Bilgin, Mehmet Emin Avşar
4	Design and development of the database.	Can Tücer	Ruşen Ali Yılmaz, Uygar Bilgin
5	Development of the general backend.	Uygar Bilgin	Göktuğ Ozan Demirtaş, Ruşen Ali Yılmaz
6	Development of the content generation module.	Göktuğ Ozan Demirtaş	Uygar Bilgin, Mehmet Emin Avşar
7	Development of the content categorization module.	Uygar Bilgin	Göktuğ Ozan Demirtaş, Mehmet Emin Avşar
8	Development of the streaming module.	Mehmet Emin Avşar	Ruşen Ali Yılmaz, Uygar Bilgin
9	Development of the user interfaces.	Ruşen Ali Yılmaz	Göktuğ Ozan Demirtaş, Can Tücer

As the majority of the packages require a long time period to be fully completed, we will work on many packages concurrently, and due to required polishing (i.e., testing & debugging edge case problems, maximizing efficiency, adding non-listed features, etc.), the majority of work packages will only be completely done when we start preparing the high-level design report. Until then, for deliverables and status reports, we will use the most up-to-date version of each implementation. Details of each listed work package can be observed within the tables below:

TABLE IV
Details for work package 1.

WP1: Finalization of project scope, features, and technical details.	
Start Date: 18.10.2025	**End Date:** 21.11.2025
Leader: Mehmet Emin Avşar	**Members:** All members
Objective: To decide on the important details of our project.	
Tasks: 1. Determining problem and solution scopes with target users and environments. 2. Determining functional & non-functional requirements. 3. Determining the technology stack & system architecture. 4. Determining the work packages and project timeline. 5. Developing the project website.	
Deliverables: 1. Project Specification Document 2. Analysis and Requirements Report	

TABLE V
Details for work package 2.

WP2: Completion of UI and interaction designs.	
Start Date: 21.11.2025	**End Date:** Late December
Leader: Ruşen Ali Yılmaz	**Members:** Göktuğ Ozan Demirtaş, Can Tücer
Objective: To have a design to follow during front-end development.	
Tasks: 1. Determining UI page structure & transitions. 2. Preparing UI of pages using Figma.	
Deliverables: 1. Demo 2. High-Level Design Report	

TABLE VI
Details for work package 3.

WP3: Development of the data scraping algorithm.	
Start Date: 21.11.2025	**End Date:** Late December
Leader: Can Tücer	**Members:** Uygar Bilgin, Mehmet Emin Avşar
Objective: To collect relevant literature sources for better content generation.	
Tasks: 1. Determining valid sources to take content from. 2. Researching data scraping strategies. 3. Implementing and running the algorithm.	
Deliverables: 1. Analysis and Requirements Report 2. Demo 3. High-Level Design Report	

TABLE VII
Details for work package 4.

WP4: Design and development of the database.	
Start Date: 01.12.2025	**End Date:** Early February
Leader: Can Tücer	**Members:** Ruşen Ali Yılmaz, Uygar Bilgin
Objective: To implement database services.	
Tasks: 1. Determining the content storage structure and object relations. 2. Creating database tables and validation constraints.	
Deliverables: 1. Demo 2. High-Level Design Report	

TABLE VIII
Details for work package 5.

WP5: Development of the general backend.	
Start Date: 01.12.2025	**End Date:** Early May
Leader: Uygar Bilgin	**Members:** Göktuğ Ozan Demirtaş, Ruşen Ali Yılmaz
Objective: To implement essential systems like authorization and state management.	
Tasks: 1. Implementing connections with databases and the content generation module. 2. Implementing REST API connections. 3. Implementing the authorization service. 4. Implementing state management services.	
Deliverables: 1. Demo 2. High-Level Design Report	

TABLE IX
Details for work package 6.

WP6: Development of the content generation module.	
Start Date: 09.12.2025	**End Date:** Early May
Leader: Göktuğ Ozan Demirtaş	**Members:** Uygar Bilgin, Mehmet Emin Avşar
Objective: To implement on-demand content generation via LLM.	
Tasks: 1. Deciding on the model and prompt to be used. 2. Implementing the LLM and API frameworks. 3. Fine-tuning the model to improve performance. 4. Implementing filtering modules to prevent the generation of dangerous content.	
Deliverables: 1. Demo 2. High-Level Design Report	

TABLE X
Details for work package 7.

WP7: Development of the content categorization module.	
Start Date: 09.12.2025	**End Date:** Early May
Leader: Uygar Bilgin	**Members:** Göktuğ Ozan Demirtaş, Mehmet Emin Avşar
Objective: To categorize scraped and cached contents based on language, grammar structures, and theme.	
Tasks: 1. Determining the language detection algorithm. 2. Determining the scope of grammar structures to detect. 3. Implementing the grammar structure detection algorithm. 4. Implementing the theme & key vocabulary words detection algorithm.	
Deliverables: 1. Demo 2. High-Level Design Report	

TABLE XI
Details for work package 8.

WP8: Development of the streaming module.	
Start Date: 09.12.2025	**End Date:** Late February
Leader: Mehmet Emin Avşar	**Members:** Ruşen Ali Yılmaz, Uygar Bilgin
Objective: To create a module that streams audio to the front end.	
Tasks: 1. Researching audio streaming and caching strategies. 2. Implementing the text-to-speech audio generation module. 3. Implementing the continuous audio streaming module. 4. Implementing the audio receiver logic into the frontend.	
Deliverables: 1. Demo 2. High-Level Design Report	

TABLE XII
Details for work package 9.

WP9: Development of the user interfaces.	
Start Date: 01.12.2025	**End Date:** Early May
Leader: Ruşen Ali Yılmaz	**Members:** Göktuğ Ozan Demirtaş, Can Tücer
Objective: To develop the application interface.	
Tasks: 1. Studying prepared designs. 2. Implementing the interface. 3. Applying best practices to improve usability and performance.	
Deliverables: 1. Demo 2. High-Level Design Report	

As stated before, due to the complexity of these packages, many will be going on concurrently until we prepare the final deliverable. Below is a Gantt chart showing our planned timeline.

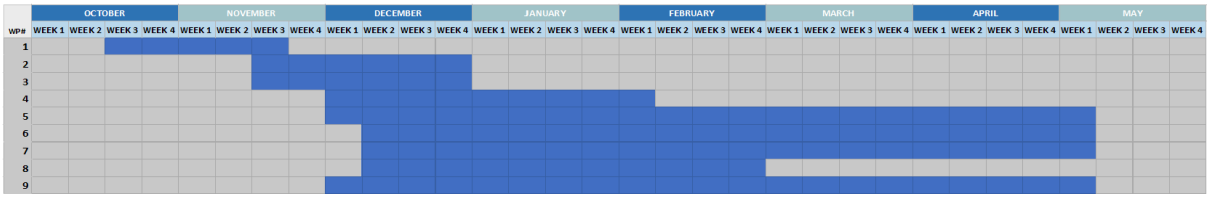

Fig. 13. Project plan Gantt chart.

3.4. Ensuring Proper Teamwork

In order to ensure proper teamwork, regular meetings will be held before upcoming deadlines. During these meetings, we will start by defining our broad goals. Then these broad goals will be broken down into their atomic constituents, and these atomic tasks will be assigned to specific team members. The allocation of tasks will be on a fairness and skill basis – those who are more suited towards one skill will be allocated related tasks, while keeping the workload of various members, as much as possible, uniform. For example, the preparation of this report has been broken down into sections, and writing each section has been assigned to a specific team member with a specific deadline. Clarity in these task assignments creates an accountable environment whereby everyone is incentivized to do their part.

3.5. Ethics and Professional Responsibilities

Ethical responsibilities can be viewed both from the perspective of inner-team relations and from the perspective of development. It is the responsibility of everyone in our team to ensure that a healthy work environment is maintained. We shall always conduct ourselves respectfully to each other and make sure to fulfill our duties as they are given to us. Every one of our peers in the team must be treated equally and responsibilities must be distributed fairly.

When we look at the development side, we see that the project will make heavy use of already existing content and AI generation. As developers we recognize the importance of ownership and copyright of any published work. To populate our dataset using already existing work, or to use code readily available on the internet, we will always check licenses and only make use of what we are allowed to according to their clauses. For AI generated content, we claim no ownership and consider any generated text available to the public.

Table XIII
Details for Ethical Considerations

Ethical Consideration	**Effect (**0: None, 10: Extremely**)**	**Mitigation**
Ownership of Content	10	Publicly available text whose copyright is expired will be used
Public Health	0	No Effect
Privacy	7	User data will be protected with Row Level Security
Misrepresenting Languages	8	LLM Orchestration will have a separate review stage to weed out inaccurate or offensive generations

Expensiveness	7	Generation will be used as a fallback mechanism and caching will be utilized

3.6. Planning for New Knowledge and Learning Strategies

All team members, being fourth year Bilkent CS students, are properly acquainted with strategies for learning new knowledge. Each team member has their own idiosyncratic way of acquiring new knowledge and their academic success up to this point proves these strategies' effectiveness. Therefore, instead of imposing a single way of skill acquisition strategy, we will allow each team member to utilize their own strategies and arrange the deadlines such that there is proper time for learning.

4. References

[1] Open-Spaced-Repetition, “ABC of FSRS,” GitHub, https://github.com/open-spaced-repetition/fsrs4anki/wiki/ABC-of-FSRS (accessed Dec. 13, 2025).

[2] D. W. Price et al., “The effect of spaced repetition on learning and knowledge transfer in a large cohort of practicing physicians,” Academic Medicine, vol. 100, no. 1, pp. 94–102, Sep. 2024. doi:10.1097/acm.0000000000005856

[3] Taalhammer, “Taalhammer | Language learning that works. Speak with precision and fluency,” taalhammer.com. https://www.taalhammer.com (accessed Dec. 12, 2025).

[4] Lenguia, “Lenguia | Learn Languages with Comprehensible Input,” lenguia.com. https://www.lenguia.com (accessed Dec. 12, 2025).

[5] Beelinguapp, “Beelinguapp | Learn languages with music & audiobooks,” beelinguapp.com. https://beelinguapp.com (accessed Dec. 12, 2025).

[6] Anki, “Anki - powerful intelligent flashcards,” apps.ankiweb.net. https://apps.ankiweb.net (accessed Dec. 12, 2025).