



Egyptian E-Learning University

Faculty of Computers & Information Technology

Graduation Project 2025

Language Learning Web Application: LinguaZone

By:

| | |
|---|---------------|
| Name: Omar Khaled Mohamed Sobaih | ID: 21-02159 |
| Name: Youssef Mohamed Youssef | ID: 21-01750 |
| Name: Ahmed Ali Saleh | ID: 21-00353 |
| Name: Youssef Medhat Shawky | ID: 21-021732 |
| Name: Engy Youssef Mored | ID: 21-00631 |
| Name: Bassant Basem Kamel | ID: 21-00563 |
| Name: Merna Wageh Thabet | ID: 21-01101 |

Under Supervision of:

- **Dr. Hanaa Bayomi**
- **Eng. Amany Abd El-Motteleb**

Abstract

This graduation project introduces *LinguaZone*, an innovative web and mobile application conceived to transform the traditional paradigm of language acquisition. At its heart, *LinguaZone* addresses the critical need for a more dynamic, personalized, and engaging digital learning environment. The core idea behind *LinguaZone* is to empower language learners with immediate, intelligent feedback that is often lacking in conventional platforms, thereby fostering faster progress and greater confidence.

The project envisions a comprehensive ecosystem where users can immerse themselves in structured lessons and interactive quizzes, all designed to make the learning journey intuitive and effective. What sets ***LinguaZone*** apart is its groundbreaking integration of artificial intelligence directly into the learning experience. This includes a sophisticated AI module capable of analyzing a user's spoken words to provide real-time grammatical feedback, guiding them towards native-like articulation. Concurrently, another intelligent component offers instant, nuanced grammar correction for written exercises, helping learners internalize correct sentence structures and usage. By bringing these advanced capabilities to a user's fingertips, ***LinguaZone*** aims to simulate the personalized attention of a human tutor within a flexible, self-paced digital format. This holistic approach promises to significantly enhance a learner's fluency and accuracy, offering a truly responsive and adaptive educational tool.

Acknowledgments

All praise and gratitude are due to Allah, the Almighty, for His boundless blessings, guidance, and strength that enabled us to successfully embark upon and complete this significant academic endeavor.

We extend our deepest and most sincere appreciation to our esteemed supervisor, **Dr. Hanna Bayomi**, for her unparalleled dedication, insightful guidance, and unwavering support throughout every phase of the LinguaZone project. Her profound expertise, meticulous review, and thoughtful feedback were not merely instrumental in shaping the technical direction of our work, but also pivotal in refining our problem-solving methodologies and fostering our growth as aspiring professionals. Her commitment to excellence served as a constant source of inspiration.

Our heartfelt thanks are also extended to **Eng. Amany Abd El-Motteleb** for her valuable assistance, encouragement, and practical insights, which significantly contributed to the smooth execution of various project components.

Furthermore, we are profoundly grateful to the **Faculty of Computers & Information Technology at the Egyptian E-Learning University (EELU)** for providing us with a stimulating academic environment, state-of-the-art resources, and the foundational knowledge essential for undertaking a project of this scope.

Contents

| | |
|--|-----------|
| Abstract | 2 |
| Acknowledgments..... | 3 |
| Introduction..... | 11 |
| 1.1 Background and motivation for the project. | 13 |
| 1.2 Importance of the problem being addressed. | 14 |
| 1.3 Problem Statement..... | 16 |
| 1.3.1 Problem | 16 |
| 1.3.2 Justification | 17 |
| 1.4 Objectives..... | 18 |
| 1.4.1 Main Objective | 18 |
| 1.4.2 Specific Objectives: | 18 |
| 1.5 Brief overview of the proposed solution..... | 20 |
| Literature Review / Related Work | 21 |
| 2.1 Research and technologies related to our project. | 22 |
| 2.1.1 Duolingo | 22 |
| 2.1.2 LearnMatch | 24 |
| 2.2 Gaps in current solutions that our project aims to fill. | 26 |
| 2.3 Summary..... | 28 |
| Proposed system..... | 29 |
| 3.1 Approach used to solve the problem | 30 |
| 3.1.2 Key practices within our Agile framework included:..... | 31 |
| 3.1.3 User-Centered Design (UCD): | 32 |
| 3.1.4 Modular and Scalable Architecture: | 33 |

| | |
|--|----|
| 3.1.5 Leveraging AI for Intelligent Feedback: | 34 |
| 3.2 System architecture (diagrams preferred: UML, flowcharts, ER diagrams, etc.) | 35 |
| 3.2.1 Overall System Architecture Diagram (Conceptual Flowchart/Component Diagram): | 35 |
| 3.2.1.1 Flow of Interaction: | 37 |
| 3.2.2 Component Diagram (illustrating software components): | 37 |
| 3.2.3 Entity-Relationship Diagram (ERD): | 38 |
| 3.2.3.1 Entities (Tables) and Their Attributes | 39 |
| 3.2.3.2 Relationships Between Entities | 42 |
| 3.2.3.3 Purpose and Design Considerations | 43 |
| 3.2.4 Sequence Diagram: | 44 |
| 3.2.4.1 User Use Cases | 45 |
| 3.2.4.1.1 Sign-up/Login | 45 |
| 3.2.4.1.2 View Lessons | 45 |
| 3.2.4.1.3 Take Quiz | 45 |
| 3.2.4.1.4 Track Progress..... | 45 |
| 3.2.4.1.5 Receive Feedback | 45 |
| 3.2.4.2 Admin Use Cases | 46 |
| 3.2.4.2.1 Create Lesson..... | 46 |
| 3.2.4.2.2 Manage Users..... | 46 |
| 3.2.4.2.3 View Analytics | 46 |
| 3.2.4.3 AI System Use Cases | 46 |
| 3.2.4.3.1 Provide Feedback..... | 46 |
| 3.2.4.3.2 Recommend Lessons | 46 |
| 3.2 Algorithms or frameworks used. | 47 |

| | |
|---|----|
| 3.3.1 Frontend Development (Flutter): | 47 |
| 3.3.2 Backend Development (Flask & Python Ecosystem): | 47 |
| 3.3.3 Artificial Intelligence / Machine Learning (Python Libraries): | 48 |
| 3.3.4 Database Management: | 50 |
| Implementation | 51 |
| 4.1 Technologies, tools, and programming languages used. | 52 |
| 4.1.1 Programming Languages | 52 |
| 4.1.2 Frameworks and Libraries | 53 |
| 4.1.3 Tools and Environment | 54 |
| 4.2 Key components/modules of the system. | 56 |
| 4.2.1 Frontend | 56 |
| 4.2.1.1 Frontend (Website) | 57 |
| 4.2.1.1.1 Navigation Bar (Header)..... | 57 |
| 4.2.1.1.2 Matching Area / Game Interface..... | 57 |
| 4.2.1.1.3 Confidence Buttons (Low, Medium, High) | 58 |
| 4.2.1.1.4 Countdown Timer | 58 |
| 4.2.1.1.5 Visual Background Decorations..... | 59 |
| 4.2.1.1.6 Feedback and Status Messages | 59 |
| 4.2.1.2 Frontend (Application) | 61 |
| 4.2.1.2.1 Design Collection | 62 |
| 4.2.1.2.2 Code Collection | 63 |
| 4.2.2 Backend (Flask Application) | 66 |
| 4.2.2.1 API Modules | 66 |
| 4.2.2.2 API playground services | 67 |
| 4.2.2.3 Coding Architecture | 71 |

| | |
|---|-----------|
| 4.2.2.3.1 Flask App Initialization & Configuration | 71 |
| 4.2.2.3.2 Authentication Service | 72 |
| 4.2.2.3.3 Level Service | 72 |
| 4.2.2.3.4 Section Service..... | 73 |
| 4.2.2.3.5 Question Service | 73 |
| 4.2.2.3.6 Quiz Service..... | 74 |
| 4.2.2.3.7 Base Service..... | 74 |
| 4.2.2.3.8 Authentication Controller..... | 75 |
| 4.2.2.3.9 Level Controller | 75 |
| 4.2.2.3.10 Section Controller | 76 |
| 4.2.2.3.11 Question Controller..... | 76 |
| 4.2.2.3.12 Quiz Controller | 77 |
| 4.2.2.3.13 Base Controller | 77 |
| 4.2.3 AI & Deep-learning..... | 78 |
| 4.2.3.1 Speech Recognition : DeepSpeech | 78 |
| 4.2.3.1.1 Environment Setup | 79 |
| 4.2.3.1.1.1 Required Dependencies | 79 |
| 4.2.3.1.1.2 Model Files..... | 79 |
| 4.2.3.1.2 Implementation | 80 |
| 4.2.3.1.2.1 Audio Recording Implementation..... | 80 |
| 4.2.3.1.2.2 Speech Recognition Implementation..... | 81 |
| 4.2.3.1.2.3 Audio Verification | 82 |
| 4.2.3.1.3 Testing | 82 |
| 4.2.3.1.4 Training..... | 83 |
| 4.2.3.1.4.1 Overview of the Dataset..... | 83 |

| | |
|--|----|
| 4.2.3.1.4.1.2 Data Splits (Implicit)..... | 84 |
| 4.2.3.1.4.2 GPU Initialization | 85 |
| 4.2.3.2 Grammar Correction : BART | 86 |
| 4.2.3.2.1 Environment Setup | 86 |
| 4.2.3.2.1.1 Required Dependencies | 87 |
| 4.2.3.2.1.2 Model Files..... | 87 |
| 4.2.3.2.1 Implementation Steps | 88 |
| 4.2.3.2.3 Testing | 88 |
| 4.2.3.2.4 Training..... | 89 |
| 4.2.3.2.4.1 Overview of the Lang-8 Corpus..... | 89 |
| 4.2.3.2.4.1.1 The .m2 Format (Maximally Informative M2) | 89 |
| 4.2.3.2.4.1.2 Data Split | 89 |
| 4.2.3.2.4.2 GPU Initialization | 90 |
| 4.2.3.3 Final Integration : DeepSpeech + BART | 90 |
| 4.2.3.3.1 Architecture | 91 |
| 4.2.3.3.2 Integration of DeepSpeech and BART Models | 91 |
| 4.2.3.3.2.1 DeepSpeech Model for Audio Transcription | 91 |
| 4.2.3.3.2.2 BART Model for Grammar Correction..... | 92 |
| 4.2.3.3.2.3 Combining the Two Models into a Single Endpoint... | 92 |
| 4.2.3.3.3 Endpoints Workflow | 93 |
| 4.2.3.3.3.1 /transcribe_audio Endpoint Workflow | 93 |
| 4.2.3.3.3.2 /process_audio Endpoint Workflow | 94 |
| 4.2.3.3.4 Model Files..... | 95 |
| 4.2.3.3.5 System Requirements | 95 |

| | |
|---|-----|
| 4.2.4 Data Layer (MySQL Database) | 96 |
| 4.2.4.1 Database Schema | 96 |
| 4.2.4.2 Data Access Layer (via SQLAlchemy) | 97 |
| 4.2.4.3 Database Models (/models folder) | 97 |
| 4.2.4.3.1 User Model (user.py)..... | 98 |
| 4.2.4.3.2 Level Model (level.py) | 98 |
| 4.2.4.3.3 Section Model (section.py) | 99 |
| 4.2.4.3.4 Question Model (question.py)..... | 99 |
| 4.2.4.3.5 Quiz Model (quiz.py)..... | 100 |
| 4.3 Challenges faced and how they were resolved | 101 |
| 4.3.1 Maintaining Timer State After Page Reload | 101 |
| 4.3.2 Managing Button Active State Dynamically | 101 |
| 4.3.3 Preventing Invalid User Interaction | 101 |
| 4.3.4 Responsive Website Design for Different Screens | 102 |
| 4.3.5 UI Responsiveness for mobile devices | 102 |
| Testing & Evaluation | 103 |
| 5.1 Testing Strategies | 104 |
| 5.1.1 Unit Testing | 104 |
| 5.1.2 Integration Testing | 105 |
| 5.1.3 User Acceptance Testing (UAT) / System Testing | 106 |
| 5.2 Performance Metrics | 107 |
| 5.2.1 AI Model Performance Metrics | 107 |
| 5.2.2 Application Performance Metrics | 109 |
| 5.3 Comparison with Existing Solutions | 110 |
| Conclusion & Future Work | 112 |

| | |
|--|------------|
| 6.1 Summary of Contributions | 113 |
| 6.2 Possible Improvements or Extensions for Future Work | 115 |
| References | 117 |
| Architecture & Design Literature | 118 |
| Core Technology Documentation..... | 118 |
| Data / Framework Sources | 119 |
| DeepSpeech’s ASR Method (Listen, Attend and Spell) | 119 |
| Lang-8 Corpus & BEA-2019 Shared Task..... | 119 |
| Foundational Transformer Papers | 120 |

Chapter 1

Introduction

1.0 Introduction

In an increasingly interconnected and globalized world, the ability to communicate effectively across diverse linguistic landscapes has become an indispensable skill, both personally and professionally. Digital platforms have emerged as a cornerstone for modern language education, offering unparalleled accessibility, flexibility, and convenience that transcend the limitations of traditional classroom settings.

However, a significant challenge persists within this digital learning sphere: many prevalent solutions, despite their widespread adoption, frequently fall short in delivering comprehensive, truly personalized, and instantaneous feedback. This deficiency is particularly apparent in the nuanced and complex areas of pronunciation and intricate grammatical structures. Learners often find themselves engaged in generic exercises that offer delayed or insufficient corrections, a pedagogical gap that can profoundly impede their progress and, critically, lead to the unintended reinforcement of errors.

The genesis of the **LinguaZone** project is rooted deeply in this identified educational void, aiming to bridge the critical divide between theoretical language acquisition and its practical, real-world application through the strategic integration of advanced technological innovations.

1.1 Background and motivation for the project.

The strategic decision to develop **LinguaZone** as a comprehensive dual-platform application, encompassing both a robust web interface and a dedicated mobile application, is meticulously driven by the imperative to maximize user accessibility and foster sustained engagement across various learning contexts.

A web-based platform inherently boasts a wider audience reach, accessible from virtually any device with a web browser, circumventing the need for specific application downloads and ensuring seamless cross-platform compatibility. This universal accessibility guarantees that learners utilizing diverse operating systems can engage with the service without compatibility barriers, cultivating a unified and consistent user experience across desktops, laptops, and mobile devices. Such consistency is pivotal, enabling users to effortlessly transition between the mobile application for on-the-go practice during commutes or short breaks, and the more immersive web platform for in-depth study, content creation, or administrative tasks.

This fluidity ensures continuity in their learning progress and data synchronization. Moreover, certain advanced functionalities, including in-depth analytical dashboards for tracking complex progress metrics, and expansive content displays for detailed explanations, are often more effectively implemented on a larger web interface. The web application further provides superior agility for content management, allowing administrators to rapidly deploy new lessons, features, and improvements, ensuring the learning content remains current, relevant, and consistent.

1.2 Importance of the problem being addressed.

The problem of inadequate real-time and personalized feedback in contemporary digital language learning environments holds profound significance because it directly compromises the efficacy, efficiency, and ultimately, the success of the language acquisition process.

Language learning is fundamentally an iterative process of practice, error, and correction. When the correction loop is delayed, generic, or non-existent, learners are left to struggle with identifying and rectifying their mistakes in isolation. This often leads to a protracted development curve and a potential plateau in their proficiency levels, where progress stagnates despite continued effort.

For instance, pronunciation errors, if left unaddressed or incorrectly self-corrected, can become deeply ingrained habits—a phenomenon known as "fossilization." Such fossilized errors can severely impede effective oral communication, leading to misunderstandings, reduced intelligibility, and a significant blow to the learner's confidence in speaking the target language. Similarly, persistent grammatical inaccuracies can profoundly hamper comprehension, both by the learner and by others, limiting academic performance, professional opportunities, and overall cultural integration.

Beyond the purely linguistic aspects, the absence of immediate and intelligent feedback also bears significant psychological implications for the learner. A lack of timely correction can lead to frustration, demotivation, and a sense of isolation. When learners cannot discern why their answers are incorrect or how to improve, their intrinsic motivation to continue practicing diminishes, potentially leading to abandonment of their language learning goals.

Conversely, immediate, constructive, and personalized feedback acts as a powerful reinforcement mechanism, validating correct responses and providing clear pathways for improvement. This fosters a sense of accomplishment, builds confidence, and encourages consistent engagement, transforming a potentially discouraging process into an empowering journey. Furthermore, traditional educational methodologies or rudimentary digital tools frequently necessitate external human intervention, such as that of a dedicated tutor, to provide such nuanced feedback. This external dependency often comes with considerable financial costs, significant time investments, and is not always readily accessible to all learners. By comprehensively addressing this multifaceted problem through the integration of advanced AI, **LinguaZone** aspires to democratize access to high-quality, personalized language instruction, thereby empowering a global community of learners to achieve fluency more efficiently, effectively, and with newfound confidence.

1.3 Problem Statement

- **Clear definition of the problem your project addresses.**
- **Justification for why this problem is worth solving.**

1.3.1 Problem: Existing digital language learning platforms frequently exhibit a critical deficiency in providing integrated, real-time, and personalized feedback mechanisms for essential linguistic competencies, specifically pronunciation and grammar. This pervasive shortcoming often compels learners to either depend on a fragmented assortment of external tools, requiring them to switch contexts and consolidate disparate pieces of information, or to continue their learning journey without the immediate benefit of corrective guidance.

Consequently, this absence of timely, targeted intervention impedes their natural progression through the learning stages and, in many cases, inadvertently reinforces erroneous linguistic patterns, which become increasingly difficult to rectify as they become ingrained. The current state of digital language education thus presents a significant barrier to achieving genuine fluency and accuracy for a large population of self-directed learners.

1.3.2 Justification: The imperative to solve this problem is of paramount importance due to its direct and profound impact on the effectiveness of language education and learner outcomes. Empirical research and pedagogical theories consistently demonstrate that immediate, intelligent, and contextually rich feedback is a cornerstone of effective skill acquisition, including language learning. Such feedback mechanisms have been consistently shown to significantly accelerate the pace of language acquisition, substantially enhance accuracy by preventing the fossilization of errors, and notably bolster learner confidence by providing clear guidance and validation.

By addressing this critical void, **LinguaZone** aims to revolutionize the digital language learning landscape. The provision of such a sophisticated, AI-driven system unequivocally makes language learning a more effective, profoundly engaging, and broadly accessible endeavor for a wider global demographic, ultimately fostering a more robust, linguistically capable, and interconnected global communication landscape that benefits individuals, industries, and international relations alike.

1.4 Objectives

- **Main Objective:** The primary goal of the project.
- **Specific Objectives:** Breakdown of tasks required to achieve the main goal.

1.4.1 Main Objective: To conceptualize, design, develop, and successfully deploy a comprehensive, cutting-edge AI-enhanced web and mobile application for language learning, named **LinguaZone**, that delivers meticulously structured lessons, highly interactive quizzes, and seamlessly integrated real-time capabilities for pronunciation assessment and sophisticated grammar correction, thereby significantly improving learner proficiency and engagement.

1.4.2 Specific Objectives:

- **To conduct thorough market research and a comprehensive comparative analysis of existing digital language learning platforms:** This objective is crucial for identifying current market gaps, discerning effective pedagogical approaches, uncovering best practices, and precisely ascertaining the unmet user needs that **LinguaZone** aims to address, ensuring its competitive differentiation and relevance.
- **To architect and implement a robust, highly scalable backend Application Programming Interface (API):** This involves creating a secure and efficient server-side infrastructure capable of reliably managing vast amounts of user profiles, dynamic lesson content, complex interactive quiz data, and seamless integration with AI models, forming the backbone of the application.

- **To develop an intuitive and user-friendly frontend web application alongside a seamlessly integrated cross-platform mobile application:**
This objective focuses on crafting highly responsive and aesthetically pleasing user interfaces that provide a consistent, engaging, and accessible learning experience across various devices, from desktops to smartphones, ensuring optimal usability.
- **To integrate and meticulously fine-tune an AI model for accurate real-time transcription and intelligent, precise pronunciation assessment:**
This involves training the model on diverse datasets to recognize a wide range of accents and speech patterns, providing learners with immediate, actionable feedback on their spoken language, thereby enhancing their phonetic accuracy.
- **To execute extensive and rigorous testing protocols, encompassing unit testing, integration testing, system testing, and user acceptance testing:**
This final objective is vital for validating the application's core functionality, ensuring its performance robustness under various loads, confirming its security integrity against vulnerabilities, and verifying its overall usability from the perspective of target users, thereby ensuring a high-quality product.

1.5 Brief overview of the proposed solution.

The proposed solution, **LinguaZone**, represents an innovative and comprehensive digital language learning platform accessible through two distinct yet interconnected interfaces: a feature-rich web application and a dedicated, responsive mobile application. At its core, **LinguaZone** offers a dynamic and rich learning environment meticulously structured around engaging lessons, which are further enhanced with diverse multimedia content including textual explanations, illustrative images, and authentic audio recordings. These lessons are seamlessly complemented by interactive quizzes designed not only to reinforce understanding but also to actively test comprehension and application of learned concepts. The true distinguishing characteristic and unique selling proposition of **LinguaZone** reside in its pioneering integration of artificial intelligence features, which provide a level of personalized feedback previously unattainable in mainstream digital language learning.

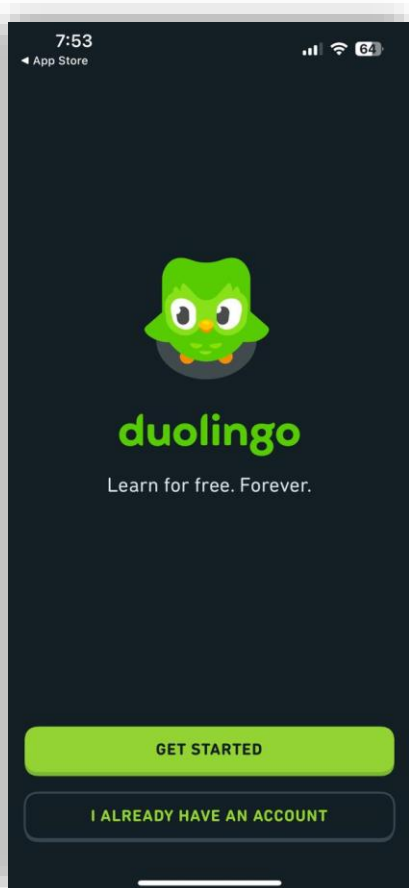
Chapter 2

Literature Review / Related Work

2.1 Research and technologies related to our project.

The digital language learning market is robust and rapidly expanding, characterized by a diverse array of platforms employing various pedagogical approaches and technological integrations. Understanding these existing solutions is crucial for identifying opportunities for innovation.

2.1.1 Duolingo: As one of the most widely adopted language learning applications globally, Duolingo has significantly popularized gamified learning.

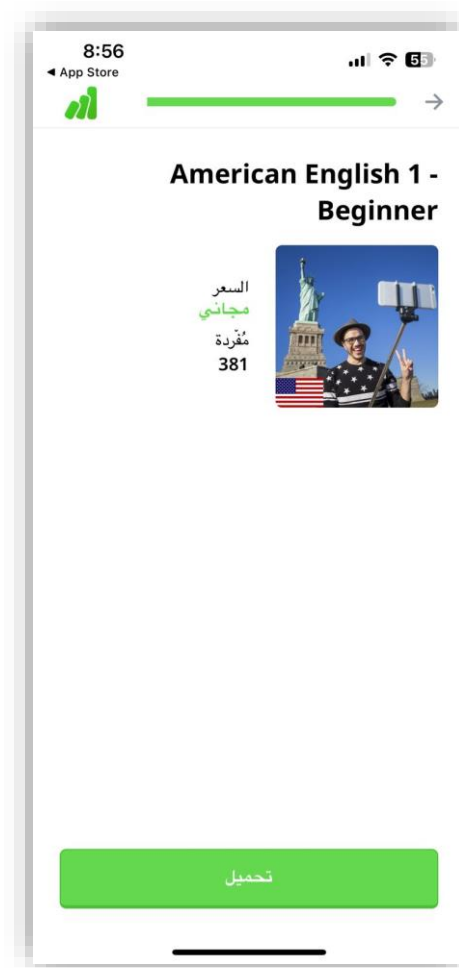
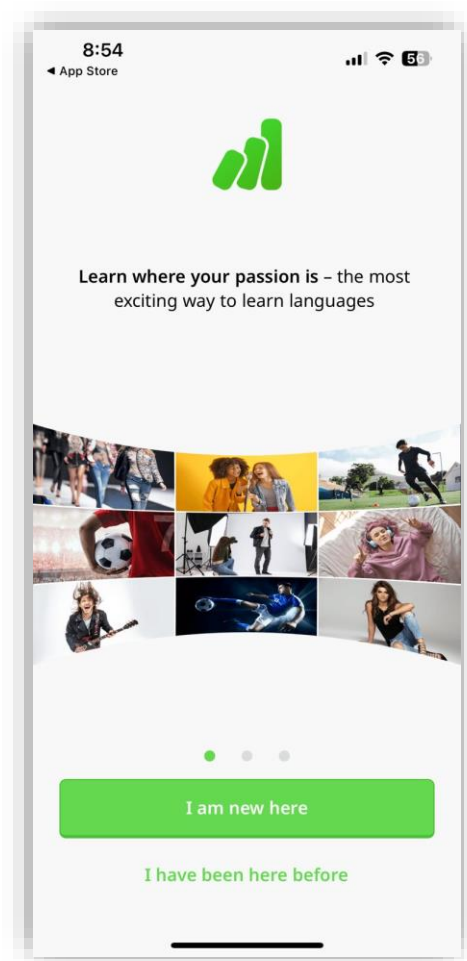


Its approach segments language acquisition into bite-sized lessons, utilizing interactive exercises such as multiple-choice questions, translation tasks, and basic listening and speaking drills.

Research indicates that Duolingo excels in initial vocabulary acquisition and maintaining user engagement through its intuitive user interface, progress tracking, and reward systems. Its gamified structure, coupled with features like streak maintenance and leaderboards, effectively motivates consistent interaction. Duolingo also customizes the initial learning experience based on user preferences.

However, while Duolingo offers speech recognition features that allow users to practice speaking, its primary focus is on basic word recognition for lesson completion, and it provides limited feedback on grammatical nuances or complex sentence structures derived from spoken input. Its feedback mechanisms on accuracy are often binary and lack the in-depth diagnostic explanations required for comprehensive grammatical improvement.

2.1.2 LearnMatch: LearnMatch distinguishes itself with its focus on dynamic and engaging vocabulary acquisition, often through interactive games and challenges tailored to specific user needs or professional contexts.



Noted for its user-friendly interface with a clean and modern design, LearnMatch customizes its courses based on user preferences and learning goals. It offers a variety of engaging exercise types including matching, fill-in-the-blanks, and multiple-choice questions, which contribute to a more active learning experience.

The platform also integrates aspects of gamification and encourages competitive learning among users. While effective for expanding vocabulary and reinforcing basic sentence structures, LearnMatch, similar to Duolingo, primarily focuses on vocabulary and general comprehension rather than analytical feedback on grammatical errors, particularly those that might arise from converting speech to text.

Its feedback mechanisms are predominantly based on correctness verification rather than providing detailed explanations for mistakes or guiding users on how to improve specific linguistic components.

Beyond these prominent examples, the market also includes platforms like Rosetta Stone, emphasizing immersion, and various tutor-matching services (e.g., italki, Preply) that offer human-led instruction.

While human tutors undoubtedly provide personalized feedback, they come with significant costs and scheduling constraints, limiting accessibility.

Other tools focus on specific aspects, such as flashcard apps for vocabulary or basic translation utilities. While advancements in speech-to-text (STT) technologies, like those leveraging deep learning architectures, have enabled accurate transcription, and Natural Language Processing (NLP) models, such as sequence-to-sequence transformers, have shown promise in grammar error correction, their comprehensive and actionable integration into accessible language learning platforms to provide grammatical feedback from spoken input remains an area ripe for innovation.

2.2 Gaps in current solutions that our project aims to fill.

Based on the review of existing language learning platforms and related technologies, several critical gaps have been identified that limit the effectiveness and depth of the learning experience:

Firstly, a significant deficiency exists in providing intelligent, context-aware grammar correction, especially when derived from user speech. While many platforms check written input, few seamlessly integrate accurate speech-to-text transcription with robust grammatical analysis. This means learners practicing speaking often don't receive detailed, actionable feedback on the grammatical correctness of their spoken sentences, only if the words were recognized. This hinders the development of accurate spoken grammar.

Secondly, the fragmentation of specialized tools forces learners to switch between multiple applications (e.g., one for lessons, another for basic speech recognition, a third for separate grammar checking, and yet another for vocabulary). This disjointed experience disrupts the learning flow, creates unnecessary cognitive load, and diminishes overall engagement and efficiency. A truly integrated solution that provides comprehensive learning and feedback within a single, cohesive platform is largely absent from the mainstream market.

Finally, while gamification undeniably enhances engagement, the core pedagogical efficacy can be undermined if the feedback loop isn't sufficiently robust and adaptive. Current solutions tend to prioritize lesson completion over genuine linguistic mastery. They often fail to adapt lesson paths effectively based on detailed, AI-driven analysis of a user's specific weaknesses in constructing grammatically correct sentences, particularly from their spoken attempts.

2.3 Summary

This literature review highlights a dynamic yet incomplete landscape within digital language learning. While existing platforms excel in areas like gamification, basic vocabulary acquisition, and broad accessibility, they consistently fall short in delivering truly personalized, real-time, and in-depth feedback on grammatical accuracy, especially when considering spoken input.

The identified gaps underscore the significant opportunity for LinguaZone. By integrating advanced, fine-tuned Models for accurate speech-to-text transcription—serving as a critical prerequisite for analysis and nuanced grammar correction applied to both transcribed and typed text, LinguaZone is uniquely positioned to fill these critical voids.

This comprehensive approach, focusing on intelligent, actionable grammatical feedback derived from diverse input methods, directly addresses the limitations of current solutions, promising a more effective, engaging, and transformative language learning experience. The insights gained from this review have critically informed the design and core functionalities of our proposed system, ensuring it is built upon a solid understanding of both established best practices and unmet user needs in the pursuit of genuine linguistic mastery.

Chapter 3

Proposed system

3.1 Approach used to solve the problem

The development of LinguaZone employed a multi-faceted approach, grounded in modern software engineering principles and an understanding of effective pedagogical strategies. Our primary goal was to create an integrated, intelligent, and user-centric platform that addresses the limitations of existing language learning solutions, specifically regarding automated and personalized grammatical feedback derived from diverse input methods, including speech.

3.1.1 Agile Development Methodology: We adopted an Agile development methodology, favoring iterative cycles and continuous feedback over a rigid, linear process.

This approach was crucial given the complexity of integrating advanced AI models and the dynamic nature of user requirements in an educational application. Sprints allowed us to develop features incrementally, test functionalities, and gather feedback, enabling rapid adaptation and refinement.

This iterative model facilitated early identification of potential challenges, such as the accuracy of speech transcription in varied environments or the nuanced understanding required for grammatical error correction, allowing for timely adjustments and optimization.

3.1.2 Key practices within our Agile framework included:

- **Sprint Planning:** Defining clear, achievable goals for each development iteration.
- **Daily Stand-ups:** Facilitating communication and problem-solving within the development team.
- **Continuous Integration:** Regularly merging code changes to detect and resolve integration issues early.
- **User Stories:** Translating user needs into actionable development tasks, ensuring the final product aligns with learner expectations.
- **Retrospectives:** Regularly reviewing processes to identify areas for improvement in subsequent sprints.

This methodology ensured that development remained responsive to both technical complexities and user needs, leading to a more robust and relevant final product.

3.1.3 User-Centered Design (UCD): A strong emphasis was placed on User-Centered Design (UCD) principles. Understanding the diverse needs, learning styles, and technical proficiencies of language learners was paramount. This involved:

- **Persona Development:** Creating detailed profiles of target users to guide design decisions.
- **User Flows:** Mapping out typical user journeys within the application to optimize navigability and task completion.
- **Wireframing and Prototyping:** Iteratively designing and testing low-fidelity and high-fidelity prototypes to gather early feedback on usability and interface intuitiveness.
- **Usability Testing:** Conducting sessions with actual users to identify pain points and areas for improvement in the user interface (UI) and user experience (UX).

The UCD approach ensured that **LinguaZone** is not only technologically advanced but also intuitively designed, engaging, and genuinely supportive of the learning process.

3.1.4 Modular and Scalable Architecture: To manage complexity, facilitate maintenance, and ensure future expandability, we adopted a modular architecture. The system was logically divided into distinct components—frontend (web and mobile), backend API, and AI processing modules—each with clearly defined responsibilities and interfaces. This modularity allows for independent development, testing, and deployment of components, minimizing interdependencies and enhancing overall system stability.

Furthermore, the architecture was designed with scalability in mind, anticipating growth in user base and data volume. This includes considerations for database optimization, efficient API endpoints, and scalable AI model serving.

3.1.5 Leveraging AI for Intelligent Feedback: The core of our solution lies in the strategic integration of Artificial Intelligence. Instead of relying on rule-based systems or simple pattern matching, we harnessed the power of deep learning models for two critical functions:

1. **Accurate Speech-to-Text Transcription:** Enabling users to speak naturally and have their words accurately converted into text, which is a prerequisite for subsequent analysis.
2. **Sophisticated Grammatical Error Correction:** Providing real-time, context-aware feedback on the grammar of both typed and transcribed input. This moves beyond merely flagging errors to suggesting precise corrections and, where possible, providing explanations, significantly enhancing the learning efficacy.

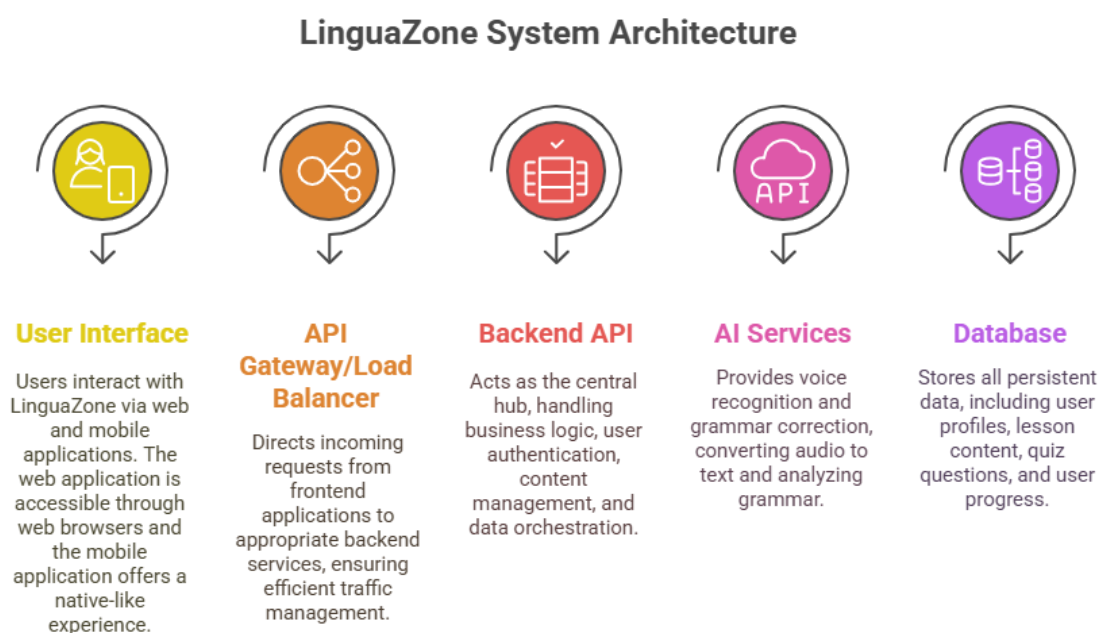
This approach ensures that **LinguaZone** offers a level of personalized and actionable linguistic feedback that is often absent in general-purpose language learning applications, directly addressing the identified problem.

3.2 System architecture (diagrams preferred: UML, flowcharts, ER diagrams, etc.).

The **LinguaZone** system is designed with a clear, three-tier architectural pattern to ensure separation of concerns, scalability, and maintainability. It comprises the **Presentation Layer (Frontend)**, the **Application Layer (Backend API and AI Services)**, and the **Data Layer (Database)**.

Below is a detailed textual description of the system's components and their interactions, which can be visually represented using various diagrams.

3.2.1 Overall System Architecture Diagram (Conceptual Flowchart/Component Diagram):









Made with  Napkin

- **User Interface (UI):** Users interact with **LinguaZone** via two primary interfaces:
 - **Web Application:** Accessible through standard web browsers, offering a rich interactive experience.
 - **Mobile Application:** Native-like experience on iOS and Android.
- **API Gateway/Load Balancer:** Directs incoming requests from frontend applications to appropriate backend services.
- **Backend API:**
 - Acts as the central hub, handling all business logic, user authentication, content management, and data orchestration.
 - Communicates with the database for data persistence and retrieval.
 - Interfaces with AI services for linguistic analysis.
- **AI Services:**
 - **Voice Recognition Service:** Dedicated service for converting user audio input (WAV/MP3) into text. It receives audio streams/files from the Backend API, processes them using the fine-tuned model, and returns the transcribed text.
 - **Grammar Correction Service:** Dedicated service for analyzing transcribed text or direct text input and providing grammatical corrections. It receives text input from the Backend API, processes it using the fine-tuned model, and returns corrected text along with suggested changes.
- **Database:** Stores all persistent data, including user profiles, lesson content, quiz questions, user progress, AI model configurations, and session information.

3.2.1.1 Flow of Interaction:

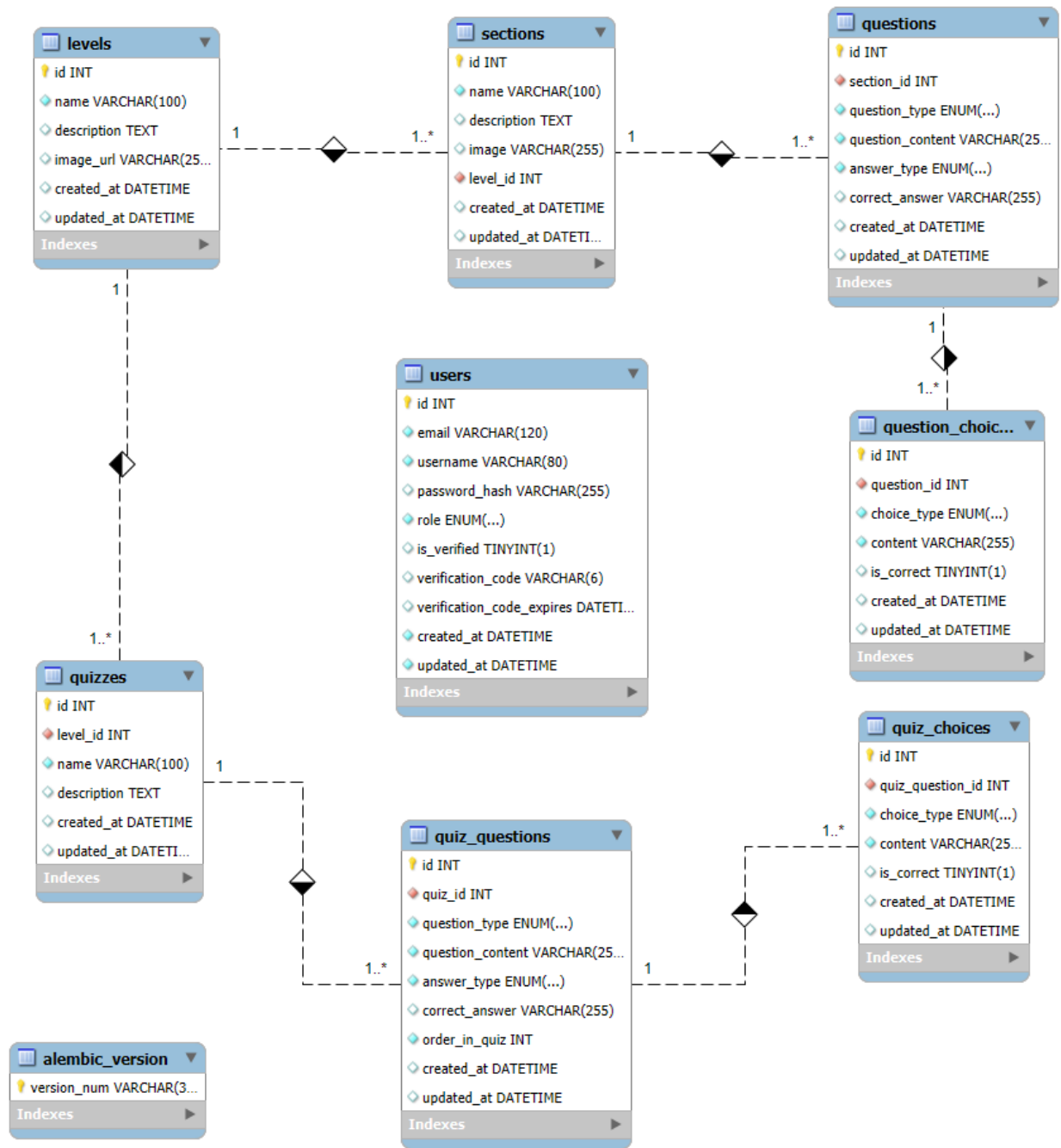
1. User interacts with Web or Mobile UI.
2. Frontend sends requests (e.g., fetch lesson, submit quiz, record audio, type text) to the Backend API.
3. Backend API processes requests:
4. AI Services perform analysis and return results to the Backend API.
5. Backend API aggregates data, updates user progress and sends responses back to the Frontend.
6. Frontend updates UI to display results

3.2.2 Component Diagram (illustrating software components):

- **User Module:** Handles user registration, login, profile management, and session management.  User Module
- **Content Management Module:** Manages lessons, quizzes, multimedia assets, and their organization.  Content Management
- **Progress Tracking Module:** Records user performance, completed lessons, quiz scores, and tracks improvements in grammatical accuracy.  Progress Tracking
- **Speech Transcription Module:** Encapsulates the AI model, handles audio input, and returns transcribed text.  Speech Transcription
- **Grammar Correction Module:** Encapsulates AI model, handles text input, and returns grammatical corrections.  Grammar Correction
- **Database Abstraction Layer (ORM):** Provides an interface for the Backend API to interact with the MySQL database, abstracting SQL queries.  Database Abstraction

Made with  Napkin

3.2.3 Entity-Relationship Diagram (ERD): An entity–relationship model (or ER model) describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between entities (instances of those entity types)



3.2.3.1 Entities (Tables) and Their Attributes

users Table

This table stores information about registered users of the LinguaZone application.

- **id** (INT): Primary Key. Unique identifier for each user.
- **email** (VARCHAR(120)): User's email address. Likely unique.
- **username** (VARCHAR(80)): User's chosen username. Likely unique.
- **password_hash** (VARCHAR(255)): Hashed password for security.
- **role** (ENUM(...)): User's role (e.g., 'learner', 'admin').
- **is_verified** (TINYINT(1)): Boolean flag indicating if the user's account is verified.
- **verification_code** (VARCHAR(6)): Code used for email verification.
- **verification_code_expires** (DATETIME): Timestamp for when the verification code expires.
- **created_at** (DATETIME): Timestamp when the user account was created.
- **updated_at** (DATETIME): Timestamp of the last update to the user's record.

levels Table

This table categorizes learning content into different proficiency levels.

- **id** (INT): Primary Key. Unique identifier for each level.
- **name** (VARCHAR(100)): Name of the level (e.g., 'Beginner', 'Intermediate').
- **description** (TEXT): A brief description of the level.
- **image_url** (VARCHAR(255)): URL to an image associated with the level.
- **created_at** (DATETIME): Timestamp when the level record was created.
- **updated_at** (DATETIME): Timestamp of the last update to the level record.

sections Table:

Sections are sub-divisions within a level, organizing lessons or content.

- **id** (INT): Primary Key. Unique identifier for each section.
- **name** (VARCHAR(100)): Name of the section.
- **description** (TEXT): A brief description of the section.
- **image** (VARCHAR(255)): URL to an image associated with the section.
- **level_id** (INT): Foreign Key referencing levels.id. Indicates which level the section belongs to.
- **created_at** (DATETIME): Timestamp when the section record was created.
- **updated_at** (DATETIME): Timestamp of the last update to the section record.

quizzes Table

This table defines quizzes within the application, often associated with a particular level.

- **id** (INT): Primary Key. Unique identifier for each quiz.
- **level_id** (INT): Foreign Key referencing levels.id. Indicates which level the quiz is associated with.
- **name** (VARCHAR(100)): Name of the quiz.
- **description** (TEXT): A brief description of the quiz.
- **created_at** (DATETIME): Timestamp when the quiz record was created.
- **updated_at** (DATETIME): Timestamp of the last update to the quiz record.

questions Table

This table stores individual questions that can be part of sections or quizzes.

- **id** (INT): Primary Key. Unique identifier for each question.
- **section_id** (INT): Foreign Key referencing sections.id. Indicates which section the question belongs to.
- **question_type** (ENUM(...)): The type of question (e.g., 'multiple_choice', 'fill_in_the_blank', 'speech_input').
- **question_content** (VARCHAR(255)): The text or content of the question.
- **answer_type** (ENUM(...)): The expected type of answer for the question.
- **correct_answer** (VARCHAR(255)): The correct answer for direct comparison questions.
- **created_at** (DATETIME): Timestamp when the question record was created.
- **updated_at** (DATETIME): Timestamp of the last update to the question record.

question_choices Table

This table stores the multiple-choice options for multiple_choice questions.

- **id** (INT): Primary Key. Unique identifier for each choice.
- **question_id** (INT): Foreign Key referencing questions.id. Links the choice to its respective question.
- **choice_type** (ENUM(...)): Type of the choice (e.g., 'text', 'image').
- **content** (VARCHAR(255)): The content of the choice option.
- **is_correct** (TINYINT(1)): Boolean flag indicating if this choice is the correct answer.
- **created_at** (DATETIME): Timestamp when the choice record was created.
- **updated_at** (DATETIME): Timestamp of the last update to the choice record.

quiz_questions Table

This is a junction table, creating a many-to-many relationship between quizzes and questions, allowing a quiz to contain multiple questions and a question to appear in multiple quizzes.

- **id** (INT): Primary Key. Unique identifier for this mapping.
- **quiz_id** (INT): Foreign Key referencing quizzes.id.
- **question_id** (INT): Foreign Key referencing questions.id.
- **question_type** (ENUM(...)): The type of question (redundant from questions table, possibly for quick reference or overriding specific quiz contexts).
- **question_content** (VARCHAR(255)): The content of the question (redundant from questions table, possibly for quick reference or overriding specific quiz contexts).
- **answer_type** (ENUM(...)): The expected type of answer (redundant from questions table, possibly for quick reference or overriding specific quiz contexts).

3.2.3.2 Relationships Between Entities

The diagram indicates the following relationships, primarily one-to-many, established through foreign keys:

- **levels to sections:** One level can have multiple sections. (One-to-Many: levels.id -> sections.level_id)
- **levels to quizzes:** One level can be associated with multiple quizzes. (One-to-Many: levels.id -> quizzes.level_id)
- **sections to questions:** One section can contain multiple questions. (One-to-Many: sections.id -> questions.section_id)
- **questions to question_choices:** One question can have multiple question_choices (relevant for multiple-choice questions). (One-to-Many: questions.id -> question_choices.question_id)

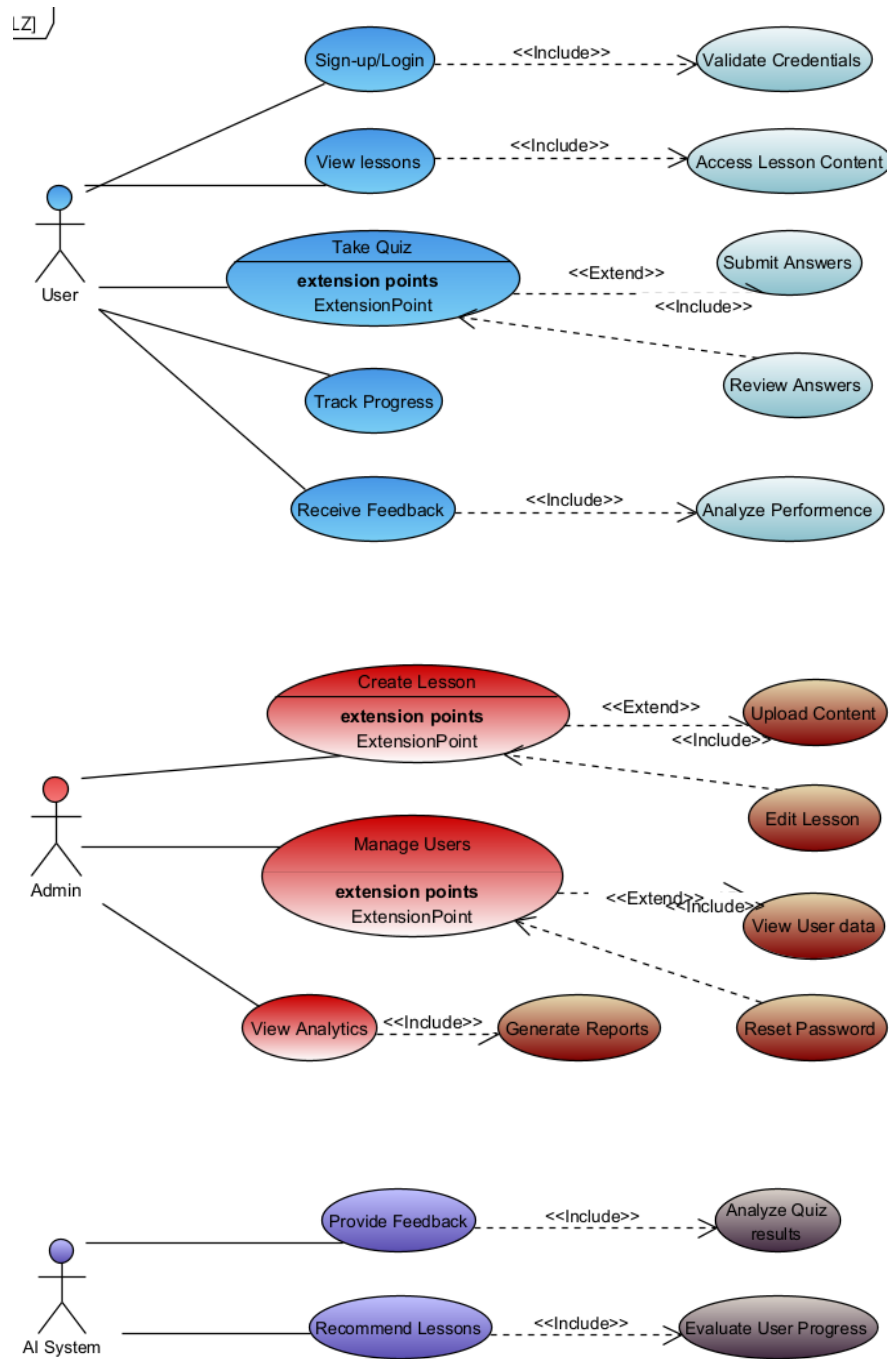
- **quizzes and questions via quiz_questions:** This forms a Many-to-Many relationship. A quiz can have many questions, and a question can belong to many quizzes.
 - (quizzes.id -> quiz_questions.quiz_id)
 - (questions.id -> quiz_questions.question_id)

3.2.3.3 Purpose and Design Considerations

This ERD represents a structured approach to managing diverse learning content and user interactions within **LinguaZone**. The design facilitates:

- **Content Organization:** Hierarchical structuring of content through levels and sections.
- **Flexible Quizzing:** Ability to create quizzes with various question types and link them to specific learning levels.
- **User Management:** Comprehensive storage for user profiles and verification details.
- **Scalability:** The relational model allows for expansion of content and user base while maintaining data integrity.
- **Extensibility:** The clear separation of concerns between entities makes it easier to add new features or content types in the future.

3.2.4 Sequence Diagram:



3.2.4.1 User Use Cases

3.2.4.1.1 Sign-up/Login

- **Description:** Enables users to register or log into the system.
- **Includes:**
 - **Validate Credentials** – Checks if the user's input credentials are valid.

3.2.4.1.2 View Lessons

- **Description:** Allows the user to view available lessons.
- **Includes:**
 - **Access Lesson Content** – Grants access to lesson materials (text, audio, images).

3.2.4.1.3 Take Quiz

- **Description:** User can take a quiz related to lessons.
- **Extension Point:**
 - **Submit Answers** – Performed upon completing the quiz.
- **Includes:**
 - **Review Answers** – Users can review the answers post-submission.

3.2.4.1.4 Track Progress

- **Description:** Users monitor their learning progress over time.

3.2.4.1.5 Receive Feedback

- **Description:** The system provides personalized feedback based on performance.
 - **Includes:**
 - **Analyze Performance** – Processes quiz and lesson data to generate feedback.
-

3.2.4.2 Admin Use Cases

3.2.4.2.1 Create Lesson

- **Description:** Enables the admin to create and manage new lessons.
- **Extension Point:**
 - Upload Content – Uploads files like text, audio, and images.
- **Includes:**
 - Edit Lesson – Modify existing lesson materials.

3.2.4.2.2 Manage Users

- **Description:** Allows admin to manage user information and permissions.
- **Extension Point:**
 - View User Data – Inspect user activity and progress.
- **Includes:**
 - Reset Password – Reset passwords for users if needed.

3.2.4.2.3 View Analytics

- **Description:** View system-wide user activity and performance.
 - **Includes:**
 - Generate Reports – Export analytics reports for administrative purposes.
-

3.2.4.3 AI System Use Cases

3.2.4.3.1 Provide Feedback

- **Description:** The AI system evaluates quiz responses to deliver targeted feedback.
- **Includes:**
 - Analyze Quiz Results – Processes user answers for correctness and patterns.

3.2.4.3.2 Recommend Lessons

- **Description:** Suggests the next lessons based on current progress.
- **Includes:**
 - Evaluate User Progress – Analyzes completion rates and success metrics.

3.2 Algorithms or frameworks used.

The successful implementation of **LinguaZone** relies on a judicious selection of modern algorithms, frameworks, and libraries, each contributing to specific aspects of the system's functionality and performance.

3.3.1 Frontend Development (Flutter):

- **Framework: Flutter** is chosen for its ability to build natively compiled applications for mobile, web, and desktop from a single codebase. This significantly reduces development time and ensures UI/UX consistency across platforms. Flutter's widget-based architecture allows for highly customizable and expressive user interfaces.
- **Google Fonts:** The Roboto font was imported using Google Fonts to improve typography and give the interface a modern, clean look.
- **Custom Visual Design:** Instead of using animation or UI frameworks, the design and animation effects were implemented entirely through custom CSS. Geometric shapes (like `.circle`, `.purple`, `.yellow`) were styled to create background depth and enhance user experience.

3.3.2 Backend Development (Flask & Python Ecosystem):

- **Framework: Flask** is a lightweight Python web framework chosen for its simplicity, flexibility, and extensibility. It allows for rapid API development, making it ideal for a RESTful API backend.
- **Database ORM: SQLAlchemy** is used as the Object-Relational Mapper (ORM) to interact with the MySQL database. SQLAlchemy provides a Pythonic way to perform database operations, enhancing code readability and reducing direct SQL query writing, while ensuring database security.

- **Authentication: Flask-JWT-Extended** for JSON Web Token (JWT) based authentication, providing a secure and stateless mechanism for user sessions.
- **Data Serialization: Marshmallow** for object serialization/deserialization, ensuring clean and consistent data exchange between the frontend and backend.
- **API Management: Flask-RESTful** or similar extensions to simplify the creation of RESTful APIs.

3.3.3 Artificial Intelligence / Machine Learning (Python Libraries):

- **Speech-to-Text Transcription:**
 - **Model: DeepSpeech (Mozilla's implementation)** is the primary algorithm used. It's an open-source end-to-end speech recognition engine trained on the Baidu DeepSpeech research paper. We utilize a fine-tuned version of this model to optimize its performance for the specific linguistic context and expected audio characteristics.
 - **Libraries:**
 - **TensorFlow** (the underlying framework for DeepSpeech) for model execution and inference.
 - **Librosa** or **Pydub** for audio preprocessing (e.g., format conversion, resampling) before feeding to the DeepSpeech model.

- **Grammar Error Correction:**

- **Model: BART (Bidirectional and Auto-Regressive Transformers)**
from the Hugging Face Transformers library is employed. BART is a denoising autoencoder for pretraining sequence-to-sequence models and is highly effective for tasks like text summarization, translation, and crucially, grammar correction. We leverage a fine-tuned BART model (e.g., a version specifically trained on grammatical error correction datasets like CoNLL-2014 or BEA-2019) to provide accurate and contextually relevant corrections.
- **Libraries:**
 - **PyTorch** or **TensorFlow** (depending on the specific BART model implementation) for model execution.
 - **Hugging Face Transformers** library for easy access to and management of pre-trained BART models and their fine-tuning.
 - **NLTK** (Natural Language Toolkit) or **SpaCy** for general text processing tasks like tokenization, sentence segmentation, and part-of-speech tagging, which can aid in preprocessing text for the BART model or in post-processing its output.

3.3.4 Database Management:

- **Database System: MySQL** is chosen as the relational database management system due to its reliability, scalability, widespread support, and robust feature set for managing structured data, including user profiles, lessons, quizzes, and progress records.

This combination of robust frameworks and advanced AI algorithms ensures that **LinguaZone** is not only technologically sound but also capable of delivering its core value proposition of intelligent, real-time grammatical feedback.

Chapter 4

Implementation

4.1 Technologies, tools, and programming languages used.

The development of **LinguaZone** leveraged a modern and robust stack of technologies, tools, and programming languages, selected for their efficiency, scalability, community support, and suitability for building an AI-enhanced, cross-platform language learning application.

4.1.1 Programming Languages

- **Python:** The primary programming language for the backend API and all Artificial Intelligence (AI) components. Python's extensive ecosystem of libraries makes it ideal for rapid development, data processing, and machine learning tasks. Its readability and active community contribute to efficient development and debugging.
- **SQL (Structured Query Language):** Used for managing and querying the relational database (MySQL). Essential for defining schema, inserting, updating, and retrieving data.
- **HTML:** HTML was used to define the structural layout of the application's web pages. Each interface is designed using semantic HTML elements to ensure clarity and accessibility.
- **CSS:** CSS was applied for visual styling, layout adjustments, and responsiveness. Custom styles were written in an external stylesheet `project.css`, and some inline styles are used where appropriate.
- **JavaScript:** JavaScript is used to enhance interactivity such as button actions and dynamic timers. The logic is mainly placed in an external script file (`script.js`) and some inline event handlers.

4.1.2 Frameworks and Libraries

- **Google Fonts:** The Roboto font was imported using Google Fonts to improve typography and give the interface a modern, clean look.
- **Custom Visual Design:** Instead of using animation or UI frameworks, the design and animation effects were implemented entirely through custom CSS. Geometric shapes (like .circle, .purple, .yellow) were styled to create background depth and enhance user experience.
- **Flutter (Frontend Framework):** A UI toolkit for building natively compiled applications for mobile (iOS, Android), web, and desktop from a single codebase. Flutter significantly accelerated development, ensured a consistent user experience across platforms, and enabled the creation of rich, interactive interfaces.
- **Flask (Backend Web Framework):** A lightweight and flexible Python web framework. Flask was chosen for its simplicity in building RESTful APIs, allowing for precise control over the application's structure and the efficient handling of HTTP requests and responses. Its modularity facilitated integration with various services, including AI models.
- **SQLAlchemy (Python ORM):** An Object-Relational Mapper that provides a high-level, Pythonic interface to interact with the MySQL database. It abstracts raw SQL, improving development speed, reducing errors, and enhancing security against SQL injection attacks.
- **Flask-JWT-Extended (Authentication):** An extension for Flask that provides support for JSON Web Tokens (JWTs). This was crucial for implementing secure, stateless user authentication and authorization for API access.

- **TensorFlow (AI Framework):** The foundational open-source machine learning framework used for executing the DeepSpeech model for speech-to-text transcription. It provides the necessary tools for numerical computation and large-scale machine learning.
- **Hugging Face Transformers (AI Library):** An essential library for working with state-of-the-art pre-trained deep learning models, including BART. It provided easy access to the BART model architecture, weights, and tokenizers, simplifying the implementation of the grammar correction service.
- **Librosa / Pydub (Audio Processing):** Python libraries used for pre-processing audio files (e.g., converting formats, resampling, normalizing) before they are fed into the DeepSpeech model, ensuring compatibility and optimal performance.
- **NLTK (Natural Language Toolkit) / SpaCy (NLP Libraries):** Employed for general natural language processing tasks such as tokenization, sentence segmentation, and potentially part-of-speech tagging, which can be useful for preparing text inputs for the BART model or processing its outputs.

4.1.3 Tools and Environment

- **Visual Studio Code (IDE):** The primary Integrated Development Environment used by the development team due to its extensive extensions, debugging capabilities, and excellent support for Python, Dart, and web development.
- **Microsoft EdgeDeveloper Tools:** Used extensively for:
 - Inspecting elements and debugging CSS layout issues
 - Testing responsiveness
 - Monitoring JavaScript console errors and network activity

- **Browser Local Storage:** The project also used the browser's localStorage API to persist data such as the timer state across page reloads.
- **MySQL (Database Management System):** A widely used open-source relational database. MySQL was chosen for its reliability, performance, and robust features for managing structured data, including user profiles, learning content, and progress tracking.
- **Postman (API Testing):** Used for testing backend API endpoints, ensuring correct functionality, response formats, and error handling during development.
- **Git / GitHub (Version Control):** For collaborative development and version control. Git managed code changes, and GitHub served as the central repository for source code, enabling team collaboration, code reviews, and project management.

This selection of technologies provided a robust and flexible foundation, enabling the development team to build a sophisticated and efficient language learning platform.

4.2 Key components/modules of the system.

The **LinguaZone** system is architected as a set of interconnected components and modules, designed to ensure modularity, maintainability, and scalability. These components align with the three-tier architecture described previously (Presentation, Application, Data Layers) and the AI services.

4.2.1 Frontend

The frontend of the **LinguaZone** application is designed to provide a consistent, intuitive, and engaging user experience across multiple platforms. Leveraging the **Flutter** framework, the development focused on building a single codebase that delivers native-like performance and a cohesive user interface on both web browsers and mobile devices (iOS and Android).

4.2.1.1 Frontend (Website)

4.2.1.1.1 Navigation Bar (Header)

A simple, fixed header that includes the project logo and navigation links to switch between sections/pages of the system.

Code Snippet (from Accessories.html)

```
<header>
  
  <nav>
    <a href="home.html">Home</a>
    <a href="Accessories.html">Accessories</a>
  </nav>
</header>
```

4.2.1.1.2 Matching Area / Game Interface

This is the main interactive area where the user is expected to match accessories with their corresponding names. It includes draggable or clickable elements and dynamically updates UI based on user actions

Code Snippet:

```
<div class="container">
  <h2>Accessories Matching</h2>
  <!-- Matching activity will be loaded here -->
</div>
```

4.2.1.1.3 Confidence Buttons (Low, Medium, High)

Custom button components that allow the user to express confidence levels in their answers. The JavaScript handles the visual activation of buttons.

Code Snippet (HTML & JS):

```
<button class="lz-confidence-btn" onclick="setConfidence(this)">Low</button>
<button class="lz-confidence-btn" onclick="setConfidence(this)">Medium</button>
<button class="lz-confidence-btn" onclick="setConfidence(this)">High</button>
```

```
function setConfidence(button) {
  const buttons = document.querySelectorAll(".lz-confidence-btn");
  buttons.forEach(btn => btn.classList.remove("active"));
  button.classList.add("active");
}
```

4.2.1.1.4 Countdown Timer

A real-time countdown timer that keeps track of how much time is left for the activity. Uses localStorage to persist time even if the page reloads.

Code Snippet:

```
let timeLeft = localStorage.getItem('timeLeft')
? parseInt(localStorage.getItem('timeLeft'))
: 15 * 60;
```

4.2.1.1.5 Visual Background Decorations

Custom shapes (like circles in different colors) are added using CSS to make the UI more lively and playful, especially for an educational environment.

Code Snippet (CSS):

```
.purple {  
  width: 70%;  
  height: 70%;  
  background-color: #28226b;  
  top: -20%;  
  left: -20%;  
}  
.circle {  
  position: absolute;  
  border-radius: 50%;  
}
```

4.2.1.1.6 Feedback and Status Messages

The system provides immediate visual feedback to users based on their performance. This is mainly done through:

- Changing **background colors** (green for correct, red for incorrect, orange for incomplete).
- Showing or hiding message areas using `.lz-message-container`.

Code Snippet (JS from script.js):

```
if (totalGreen === 5) {  
    main.style.backgroundColor = "green";  
    messageContainer.style.display = "block";  
    message.textContent = "Well done!";  
} else if (totalRed > 0) {  
    main.style.backgroundColor = "red";  
    messageContainer.style.display = "block";  
    message.textContent = "Some answers are incorrect.";  
} else {  
    main.style.backgroundColor = "orange";  
    messageContainer.style.display = "block";  
    message.textContent = "Please complete all matches.";  
}
```

Task 1

1. What is the topic of the video?

- ☐ Weather
- ☒ Shopping
- ☐ Travel

2. The video includes a conversation between two people.

- ☐ True
- ☒ False

Task 1: You got 1 out of 2 correct.

4.2.1.2 Frontend (Application)

The application includes various sections such as Courses, Listening Exercises, Reading Texts, Memory Games, and Interactive Tests. Each section is designed with a user-friendly interface and smooth navigation to enhance the learning experience. The app was fully developed using the Flutter framework, which provided a flexible and efficient development process.

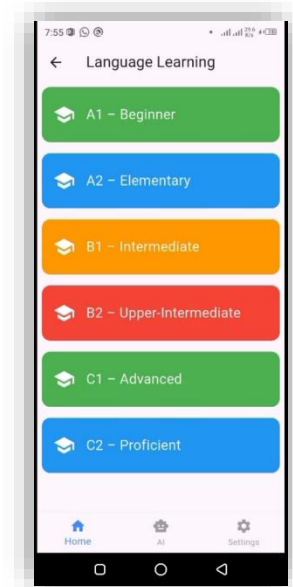
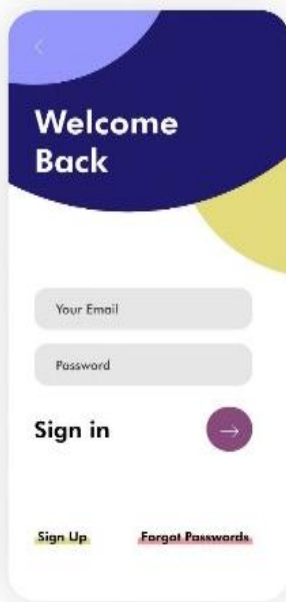
After presenting the theoretical and structural aspects of the project, this section will demonstrate the practical side of the application. It includes the actual screens designed for user interaction, as well as the code implemented to bring the features to life.

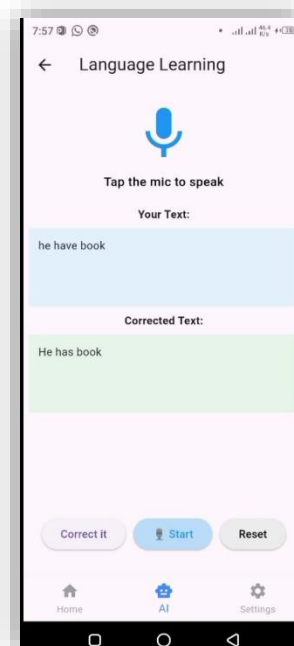
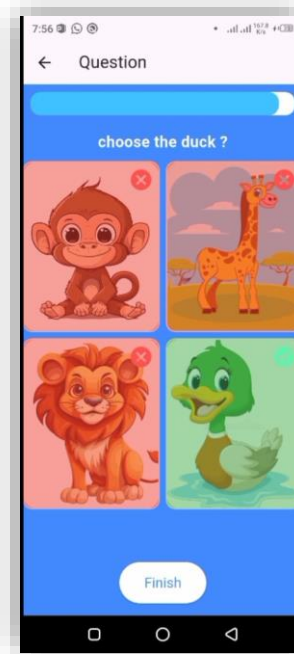
The implementation part is divided into two main sections:

- Design Collection: showcasing the visual appearance of each screen.
- Code Collection: presenting the code snippets used to build the functionalities behind those screens.

4.2.1.2.1 Design Collection

This section contains the visual design of each screen in the application. It helps illustrate how the user interacts with the app. Each design screenshot is accompanied by a short description explaining the purpose of the screen and the user's expected actions.





4.2.1.2.2 Code Collection

This section presents the implementation of different parts of the application. Each code snippet is related to a specific screen or feature and is explained briefly to show its function within the app

```
all > lib > splash > splash.dart > FlutterApp > build
72 class Splash extends StatelessWidget {
73   const Splash({super.key});
74
75   @override
76   Widget build(BuildContext context) {
77     return GestureDetector(
78       onTap: () {
79         Navigator.push(
80           context,
81           MaterialPageRoute(builder: (context) => SignUp()),
82         );
83       },
84       child: Container(
85         width: 390,
86         height: 844,
87         clipBehavior: Clip.antiAlias,
88         decoration: BoxDecoration(
89           borderRadius: BorderRadius.circular(40),
90           image: const DecorationImage(
91             image: AssetImage("assets/splash.png"),
92             fit: BoxFit.fill,
93           ), // DecorationImage
94         ), // BoxDecoration
95         child: Stack(
96           children: [
97             Positioned(
98               left: 33,
99               top: 692,
100               child: Container(
101                 width: 329,
102                 height: 49,
103                 decoration: BoxDecoration(
104                   color: const Color(0xFF4E89EF),
105                   borderRadius: BorderRadius.circular(10),
106                 ), // BoxDecoration
```

```
all > lib > sign up > sign up.dart > SignUp > build
70 class SignUp extends StatelessWidget {
71   const SignUp({super.key});
72
73   @override
74   Widget build(BuildContext context) {
75     return Scaffold(
76       body: SingleChildScrollView(
77         child: Column(
78           mainAxisAlignment: MainAxisAlignment.start,
79           children: [
80             Container(
81               width: 390,
82               height: 844,
83               clipBehavior: Clip.antiAlias,
84               decoration: BoxDecoration(
85                 color: const Color(0xFF4E89EF),
86                 shape: RoundedRectangleBorder(
87                   borderRadius: BorderRadius.circular(40),
88                 ), // RoundedRectangleBorder
89               shadows: [
90                 BoxShadow(
91                   color: const Color(0x26000000),
92                   blurRadius: 75,
93                   offset: Offset(0, 0),
94                   spreadRadius: 0,
95                 ) // BoxShadow
96               ],
97             ), // BoxDecoration
98             child: Stack(
99               children: [
100                 Positioned(
101                   left: 110.47,
102                   top: 769.29,
103                   child: Transform(
104                     transform: Matrix4.identity()
```

```
all > lib > courses > courses.dart > FlutterApp > build
70 class Courses extends StatelessWidget {
71   @override
72   Widget build(BuildContext context) {
73     return Scaffold(
74       body: SingleChildScrollView(
75         child: Column(
76           children: [
77             Container(
78               width: 390,
79               height: 772,
80               clipBehavior: Clip.antiAlias,
81               decoration: BoxDecoration(
82                 color: const Color(0xFF4E89EF),
83                 shape: RoundedRectangleBorder(
84                   borderRadius: BorderRadius.circular(40),
85                 ), // RoundedRectangleBorder
86               shadows: [
87                 BoxShadow(
88                   color: const Color(0x3F000000),
89                   blurRadius: 75,
90                   offset: Offset(0, 4),
91                   spreadRadius: 0,
92                 ) // BoxShadow
93               ],
94             ), // ShapeDecoration
95             child: Stack(
96               children: [
97                 Positioned(
98                   left: 157,
99                   top: 10,
100                   child: Container(width: 85, height: 96.76),
101                 ), // Positioned
102                 Positioned(
103                   left: 73,
104                   top: 185,
```

```
all > lib > vocabulary > vocabulary.dart > Vocabulary > build
71 class Vocabulary extends StatelessWidget {
72   @override
73   Widget build(BuildContext context) {
74     return Scaffold(
75       body: SingleChildScrollView(
76         child: Column(
77           children: [
78             Container(
79               width: 390,
80               height: 1113,
81               clipBehavior: Clip.antiAlias,
82               decoration: BoxDecoration(
83                 color: Colors.white,
84                 shape: RoundedRectangleBorder(
85                   borderRadius: BorderRadius.circular(40),
86                 ), // RoundedRectangleBorder
87               shadows: [
88                 BoxShadow(
89                   color: const Color(0x3F000000),
90                   blurRadius: 75,
91                   offset: Offset(0, 4),
92                   spreadRadius: 0,
93                 ) // BoxShadow
94               ],
95             ), // ShapeDecoration
96             child: Stack(
97               children: [
98                 Positioned(
99                   left: 331,
100                   top: 17,
101                   child: Container(
102                     width: 35,
103                     height: 35,
104                     decoration: BoxDecoration(
105                       image: DecorationImage(
```

```
languagelearning > lib > domain > repo > % Repository > sectionsById
11
12 Windsor: Refactor | Explain | Qodo Gen: Options | Test this class
13 class Repository {
14   final DioClient api;
15
16   Repository(this.api);
17
18   Windsor: Refactor | Explain | Generate Function Comment | X | Qodo Gen: Options | Test this method
19   Future<Either<Failure, LoginResponse>> userLogin(
20     String email,
21     String password,
22   ) async {
23     try {
24       final response = await api.post(
25         Apilris.login,
26         data: {'email': email, 'password': password},
27       );
28       final loginResponse = LoginResponse.fromJson(response.data);
29       final prefs = await SharedPreferences.getInstance();
30       await prefs.setString('access_token', loginResponse.accessToken);
31     } catch (e) {
32       return Right(loginResponse);
33     } on DioException catch (e) {
34       return Left(Failure(e.toString()));
35     } catch (e) {
36       return Left(Failure(e.toString()));
37     }
38   }
39
40 Windsor: Refactor | Explain | Generate Function Comment | X | Qodo Gen: Options | Test this method
41 Future<Either<Failure, RegisterResponse>> userRegister(
42   String name,
43   String email,
44   String password,
45   String role,
46 ) async {
47   try {
48     final response = await api.post(
49       Apilris.register,
50       data: {'email': email, 'password': password, 'role': 'ADMIN', 'username': name},
51     );
52     final loginResponse = RegisterResponse.fromJson(response.data);
53     final prefs = await SharedPreferences.getInstance();
54     await prefs.setString('access_token', loginResponse.accessToken);
55   } catch (e) {
56     return Right(loginResponse);
57   } on DioException catch (e) {
58     return Left(Failure(e.toString()));
59   } catch (e) {
60     return Left(Failure(e.toString()));
61   }
62 }
```

```
languagelearning > lib > features > question > imageQuestion > % imageQuestion.dart > % ImageQuestionScreen > % build
10
11 Windsor: Refactor | Explain | Qodo Gen: Options | Test this class
12 class ImageQuestionScreen extends StatelessWidget {
13   ImageQuestionScreen({super.key, required this.questions});
14   static const String routeName = '/imagequestion';
15
16   final List<QuestionModel> questions;
17
18   @override
19   Widget build(BuildContext context) {
20     return MultiBlocProvider(
21       providers: [
22         BlocProvider(create: (_) => ScoreTrackerCubit(totalQuestions: questions.length)),
23         BlocProvider(create: (_) => QuestionCubit(totalQuestions: questions.length)),
24       ],
25     );
26     child: Scaffold(
27       backgroundColor: Colors.blueAccent,
28       appBar: AppBar(title: const Text('Memory Question')),
29       body: BlocBuilder<QuestionCubit, QuestionState>(
30         builder: (context, questionState) {
31           if (questionState.currentIndex >= questions.length) {
32             return Center(
33               child: QuizCompletedDialog(
34                 totalQuestions: questions.length,
35               ), // QuizCompletedDialog
36             ); // Center
37           }
38           final question = questions[questionState.currentIndex];
39
40           return Column(
41             children: [
42               const SizedBox(height: 10),
43               QuestionsTracker(total: questions.length),
44               const SizedBox(height: 20),
45               // Display the question image
46               Container(
47                 height: 200,
48                 width: double.infinity,
49                 margin: const EdgeInsets.symmetric(horizontal: 20),
50                 decoration: BoxDecoration(
51                   borderRadius: BorderRadius.circular(12),
52                   border: Border.all(color: Colors.white, width: 2),
53                 ), // BoxDecoration
54             ],
55           );
56 }
```

```
return Scaffold(
  appBar: AppBar(
    title: const Text('Words from PDF'),
    centerTitle: true,
    backgroundColor: Colors.lightBlue[700],
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(15.0),
    ),
    toolbarHeight: 60.0,
  ),
  body: Center(
    child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Container(
        padding: const EdgeInsets.all(20.0),
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(20.0),
          boxShadow: [
            BoxShadow(
              color: Colors.grey.withOpacity(0.5),
              spreadRadius: 5,
              blurRadius: 7,
              offset: const Offset(0, 3), // changes
            ),
          ],
        ),
      ),
    ),
  ),
);
```

```
class BronzeChickenHomePage extends StatelessWidget {
  const BronzeChickenHomePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Bronze Chicken Display'),
        centerTitle: true,
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // Display the main text "BRONZE CHICKEN"
            Text(
              'BRONZE CHICKEN',
              style: TextStyle(
                fontSize: 32,
                fontWeight: FontWeight.bold,
                color: Colors.brown[700],
              ),
            ),
            const SizedBox(height: 20), // Add some spacing
            // Display the secondary text "2/15"
            Text(
              '2/15',
            ),
          ],
        ),
      ),
    );
  }
}
```

4.2.2 Backend (Flask Application)

4.2.2.1 API Modules

- **Authentication & Authorization Module:** Manages user authentication (JWT-based), validates user tokens, and controls access to different API endpoints based on user roles and permissions.
- **User Management Module:** Handles CRUD (Create, Read, Update, Delete) operations for user accounts, including profile updates and status changes.
- **Content Management Module:** Provides API endpoints for managing levels, sections, quizzes, questions, and `question_choices`. This module allows administrators to add, modify, and delete learning content.
- **User Progress Module:** Stores and retrieves user performance data, including quiz scores, lesson completion status, and detailed grammar correction results. It's responsible for persisting and providing data for the frontend dashboard.
- **AI Integration Module (Gateway):** This is a critical module that acts as an intermediary between the core Backend API and the dedicated AI microservices. It receives audio input for transcription, forwards it to the DeepSpeech Service, receives the transcribed text, then forwards this (or direct text input) to the BART Grammar Correction Service, and finally processes and returns the AI-generated feedback to the frontend. It manages the asynchronous communication with AI services.

4.2.2.2 API playground services

Authentication Endpoints (/api/auth)

- POST /api/auth/register - Register a new user.
- POST /api/auth/verify-email - Verify a user's email with a verification code.
- POST /api/auth/resend-otp - Resend the email verification OTP.
- POST /api/auth/is-user-email-found - Check if a user's email exists.
- POST /api/auth/login - Log in a user and generate JWT tokens.
- DELETE /api/auth/delete-user - Delete a user by email.
- POST /api/auth/forgot-password - Request a password reset.
- POST /api/auth/retrieve-password - Reset a user's password using a verification code.
- POST /api/auth/refresh - Refresh the access token using a refresh token.

Level Endpoints (/api/level)

- GET /api/level - Get all levels.
- POST /api/level - Create a new level.
- GET /api/level/<int:level_id> - Get a specific level by ID.
- PUT /api/level/<int:level_id> - Update a specific level by ID.
- DELETE /api/level/<int:level_id> - Delete a specific level by ID.

Section Endpoints (/api/section)

- GET /api/section - Get all sections or filter by level using [level_id](#) query parameter.
 - POST /api/section - Create a new section.
 - GET /api/section/<int:section_id> - Get a specific section by ID.
 - PUT /api/section/<int:section_id> - Update a specific section by ID.
 - DELETE /api/section/<int:section_id> - Delete a specific section by ID.
-

Question Endpoints (/api/question)

- GET /api/question - Get all questions or filter by section using [section_id](#) query parameter.
- POST /api/question - Create a new question.
- GET /api/question/<int:question_id> - Get a specific question by ID.
- PUT /api/question/<int:question_id> - Update a specific question by ID.
- DELETE /api/question/<int:question_id> - Delete a specific question by ID.
- POST /api/question/<int:question_id>/choices - Add choices to a question.

- PUT /api/question/<int:question_id>/choices/<int:choice_id> - Update a specific choice for a question.
 - DELETE /api/question/<int:question_id>/choices/<int:choice_id> - Delete a specific choice from a question.
-

Quiz Endpoints (/api/quiz)

- POST /api/quiz – Create a quiz
 - POST /api/quiz/<id>/questions – Add questions to a quiz
 - GET /api/quiz/<id> - Get a quiz with its question
 - DELETE /api/quiz/<id> - Delete a quiz
-

General Endpoints (/api/general)

- GET /api/general/<endpoint> - Get all items for a specific endpoint (e.g., [level](#), [section](#), [question](#), questionChoice).
- POST /api/general/<endpoint> - Create an item for a specific endpoint.
- GET /api/general/<endpoint>/<int:id> - Get a specific item by ID for a specific endpoint.

- PUT /api/general/<endpoint>/<int:id> - Update a specific item by ID for a specific endpoint.
 - DELETE /api/general/<endpoint>/<int:id> - Delete a specific item by ID for a specific endpoint.
 - POST /api/general/upload - Upload a file.
-

Static and Template Endpoints

- GET /static/<path:filename> - Serve static files (e.g., images, CSS, JavaScript).
- GET /templates/<template_name> - Render HTML templates (e.g., [levels.html](#)).

4.2.2.3 Coding Architecture

4.2.2.3.1 Flask App Initialization & Configuration

```
app.py
1  print("Starting the Flask app")
2
3  from app import create_app
4
5  app = create_app()
6
7  if __name__ == "__main__":
8      app.run(debug=True)
9
10
```

```
config.py
1  import os
2  from datetime import timedelta
3
4  class Config:
5      DEBUG = True
6      SECRET_KEY = os.environ.get('SECRET_KEY', 'your-secret-key-here')
7      SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://root:3125@127.0.0.1/language_learning'
8      SQLALCHEMY_TRACK_MODIFICATIONS = False
9      JWT_SECRET_KEY = os.environ.get('JWT_SECRET_KEY', 'your-jwt-secret-key')
10     JWT_ACCESS_TOKEN_EXPIRES = timedelta(minutes=600)
11     JWT_REFRESH_TOKEN_EXPIRES = timedelta(days=30)
12
13
14     # Mail Configuration
15     MAIL_SERVER = 'smtp.gmail.com'
16     MAIL_PORT = 587
17     MAIL_USE_TLS = True
18     MAIL_USERNAME = 'linguazone3125@gmail.com'
19     MAIL_PASSWORD = 'fcuojtyzjwrcpltg'
20     MAIL_DEFAULT_SENDER = 'linguazone3125@gmail.com'
21
22     UPLOAD_FOLDER = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'static', 'uploads')
23     ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'webp'}
24     MAX_CONTENT_LENGTH = 5 * 1024 * 1024 # 5MB max file size
25
26
```

4.2.2.3.2 Authentication Service

```

9 class AuthService:
10     @staticmethod
11     def register(email, password, username, role):
12         """
13         Register a new user and return access token
14         """
15         if User.query.filter_by(email=email).first():
16             raise ValueError("Email already registered")
17
18         user = User(
19             email=email,
20             username=username,
21             role=role
22         )
23         user.set_password(password)
24
25         db.session.add(user)
26         db.session.commit()
27
28         # Create access token
29         access_token = create_access_token(identity=user.id)
30
31         return {
32             'access_token': access_token,
33             'user': {
34                 'id': user.id,
35                 'email': user.email,
36                 'username': user.username,
37                 'role': user.role.value
38             }
39         }
40

```

4.2.2.3.3 Level Service

```

app > services > level_service.py
8 class LevelService:
9     def get_all(self):
10         levels = Level.query.all()
11         return [level.to_dict() for level in levels]
12
13     def get_by_id(self, level_id):
14         level = Level.query.get(level_id)
15         return level.to_dict() if level else None
16
17     def create_level(self, data, file=None):
18         try:
19             # Handle file upload if provided
20             image_url = None
21             if file:
22                 try:
23                     image_url = save_file(file, 'levels')
24                 except Exception as e:
25                     raise Exception(f"Failed to save file: {str(e)}")
26
27             level = Level(
28                 name=data['name'],
29                 description=data.get('description', ''),
30                 image_url=image_url
31             )
32             db.session.add(level)
33             db.session.commit()
34             return level.to_dict()
35         except SQLAlchemyError as e:
36             db.session.rollback()
37             raise Exception(f"Database error: {str(e)}")
38

```

4.2.2.3.4 Section Service

```
app > services > section_service.py
37 def update_section(self, section_id: int, data: Dict[str, Any], file: Optional[FileStorage] = None) -> Optional[Section]:
38     """
39     Update an existing section with optional file upload.
40     """
41     section = self.get_by_id(section_id)
42     if not section:
43         return None
44
45     if file:
46         # Delete old file if exists
47         if section.image:
48             delete_file(section.image)
49         # Save new file
50         file_path = save_file(file, 'sections')
51         data['image'] = file_path
52
53     return self.update(section_id, data)
54
55 def delete_section(self, section_id: int) -> bool:
56     """
57     Delete a section and its associated file.
58     """
59     section = self.get_by_id(section_id)
60     if not section:
61         return False
62
63     # Delete associated file if exists
64     if section.image:
65         delete_file(section.image)
66
67     return self.delete(section_id)
68
```

4.2.2.3.5 Question Service

```
app > services > question_service.py
93 # Handle choices for multiple choice questions
94 if atype == AnswerType.MULTIPLE_CHOICE:
95     if not choices_data:
96         raise BadRequest("Choices are required for multiple-choice questions.")
97
98     # --- DEBUGGING STATEMENTS START (create_question) ---
99     print(f"\n--- DEBUGGING CHOICES START (create question) ---")
100     print(f"Received choices_data: {choices_data}")
101
102     has_correct = False # Flag to ensure at least one correct choice
103
104     for idx, choice_dict in enumerate(choices_data):
105         print(f" Processing choice {idx}: {choice_dict}")
106
107         choice_content = choice_dict.get('content')
108         if not choice_content or not str(choice_content).strip():
109             raise BadRequest(f"Choice content for choice {idx} cannot be empty.")
110
111         is_correct_val = choice_dict.get('is_correct', False)
112         if isinstance(is_correct_val, str):
113             is_correct_val = is_correct_val.lower() == 'true'
114         elif isinstance(is_correct_val, (int, float)):
115             is_correct_val = bool(is_correct_val)
116
117         print(f" is_correct_val (after conversion): {is_correct_val} (type: {type(is_correct_val)})")
118
119         if is_correct_val:
120             has_correct = True
121             print(f" Choice {idx} is CORRECT! has_correct is now {has_correct}")
122         else:
123             print(f" Choice {idx} is NOT correct.")
124
125     current_choice_type = ChoiceType.TEXT # Using the imported ChoiceType enum
```

4.2.2.3.6 Quiz Service

```
app > services > quiz_service.py
223 def create_quiz_question(self, quiz_id: int, question_data: Dict[str, Any], files: Optional[Dict[str, Any]] = None) -> QuizQuestion:
224     quiz = Quiz.query.get(quiz_id)
225     if not quiz:
226         raise BadRequest(f"Quiz with ID {quiz_id} not found.")
227
228     order_in_quiz = question_data.get('order_in_quiz')
229     if order_in_quiz is None:
230         order_in_quiz = QuizQuestion.query.filter_by(quiz_id=quiz_id).count() + 1
231     else:
232         try:
233             order_in_quiz = int(order_in_quiz)
234         except ValueError:
235             raise BadRequest("Order in quiz must be an integer.")
236
237     try:
238         quiz_question = self._prepare_quiz_question_for_addition(quiz_id, question_data, order_in_quiz, files)
239
240         db.session.commit()
241         return quiz_question
242     except BadRequest as e:
243         db.session.rollback()
244         raise e
245     except SQLAlchemyError as e:
246         db.session.rollback()
247         print(f"Database error during single quiz question creation: {e}")
248         raise BadRequest(f"Database error during single quiz question creation: {str(e)}")
249     except Exception as e:
250         db.session.rollback()
251         print(f"Unexpected error during single quiz question creation: {e}")
252         raise BadRequest(f"An unexpected error occurred during single quiz question creation: {str(e)}")
```

4.2.2.3.7 Base Service

```
app > services > base_service.py
10 class BaseService:
11     """Base service class with common CRUD operations."""
12
13     def __init__(self, model_class: Type[T]):
14         """Initialize the service with a model class."""
15         self.model_class = model_class
16
17     def get_all(self) -> List[T]:
18         """Get all records from the database."""
19         return self.model_class.query.all()
20
21     def get_by_id(self, id: int) -> Optional[T]:
22         """Get a record by its ID."""
23         return self.model_class.query.get(id)
24
25     def create(self, data: Dict[str, Any]) -> T:
26         """Create a new record."""
27         instance = self.model_class(**data)
28         db.session.add(instance)
29         db.session.commit()
30         return instance
31
32     def update(self, id: int, data: Dict[str, Any]) -> Optional[T]:
33         """Update a record by its ID."""
34         instance = self.get_by_id(id)
35         if instance:
36             for key, value in data.items():
37                 setattr(instance, key, value)
38                 db.session.commit()
39         return instance
```

4.2.2.3.8 Authentication Controller

```
app > controllers > api > auth_controller.py
9 @auth_bp.route('/register', methods=['POST'])
10 def register():
11     data = request.get_json()
12
13     required_fields = ['email', 'password', 'username', 'role']
14     if not all(field in data for field in required_fields):
15         return jsonify({'error': 'Missing required fields'}), 400
16
17     try:
18         result = AuthService.register(
19             email=data['email'],
20             password=data['password'],
21             username=data['username'],
22             role=data['role']
23         )
24         return jsonify({
25             'message': 'User registered successfully',
26             **result
27         }), 201
28     except ValueError as e:
29         return jsonify({'error': str(e)}), 400
30     except RuntimeError as e:
31         return jsonify({'error': str(e)}), 500
32     except Exception as e:
33         return jsonify({'error': str(e)}), 500
```

4.2.2.3.9 Level Controller

```
app > controllers > api > level_controller.py
55 def create_level(self) -> Tuple[Dict[str, Any], int]:
56     """ Create a new level. """
57     try:
58         # Support both JSON and form-data
59         if request.is_json:
60             data = request.get_json()
61             file = None
62         else:
63             data = request.form
64             file = request.files.get('image')
65
66         if 'name' not in data:
67             return self.error_response("Name is required", status_code=400)
68
69         if file:
70             try:
71                 validate_file_upload(file)
72             except (BadRequest, FileUploadError) as e:
73                 return self.error_response(str(e), status_code=400)
74
75         level = self.service.create_level(data, file)
76         return self.success_response(
77             data=level,
78             message="Level created successfully",
79             status_code=201
80         )
81     except BadRequest as e:
82         logger.warning(f"Bad request while creating level: {str(e)}")
83         return self.error_response(str(e))
84     except Exception as e:
85         logger.error(f"Error creating level: {str(e)}")
86         return self.error_response("Failed to create level", status_code=500)
```

4.2.2.3.10 Section Controller

```
app > controllers > api > section_controller.py
10 class SectionController(BaseController):
11     """Controller for handling section-related operations."""
12
13     def __init__(self):
14         """Initialize the section controller."""
15         super().__init__('section', __name__)
16         self.service = SectionService()
17         self._register_routes()
18
19     def _register_routes(self) -> None:
20         """Register all routes for the section controller."""
21         self.blueprint.route('', methods=['GET'], strict_slashes=False)(token_required(self.get_sections))
22         self.blueprint.route('/<int:section_id>', methods=['GET'], strict_slashes=False)(token_required(self.get_section))
23         self.blueprint.route('', methods=['POST'], strict_slashes=False)(admin_required(self.create_section))
24         self.blueprint.route('/<int:section_id>', methods=['PUT'], strict_slashes=False)(admin_required(self.update_section))
25         self.blueprint.route('/<int:section_id>', methods=['DELETE'], strict_slashes=False)(admin_required(self.delete_section))
26
27     def get_sections(self) -> Tuple[Dict[str, Any], int]:
28         """Get all sections or filter by level. """
29         level_id = request.args.get('level_id', type=int)
30         sections = self.service.get_sections_by_level(level_id) if level_id else self.service.get_all()
31         return self.success_response(data=[section.to_dict() for section in sections])
32
33     def get_section(self, section_id: int) -> Tuple[Dict[str, Any], int]:
34         """Get a specific section by ID. """
35         section = self.service.get_by_id(section_id)
36         if not section:
37             return self.error_response("Section not found", status_code=404)
38         return self.success_response(data=section.to_dict())
```

4.2.2.3.11 Question Controller

```
app > controllers > api > question_controller.py
82 def update_question(self, question_id: int) -> Tuple[Dict[str, Any], int]:
83     """Update an existing question."""
84     try:
85         data = {}
86         question_file = None
87         files_for_choices = None
88
89         if request.is_json:
90             data = request.get_json()
91             # For JSON, 'choices' if present, would already be a list/dict.
92             # No files for choices expected directly in request.files for JSON.
93         else: # Handle multipart/form-data
94             data = request.form.to_dict() # Get form fields as a dict
95             # Assuming your main question media file input is named 'question_content'
96             question_file = request.files.get('question_content')
97             files_for_choices = request.files # All files in the request, for choice media
98
99         if question_file:
100             validate_file_upload(question_file)
101
102         # Pass files_for_choices to the service method
103         question = self.service.update_question(question_id, data, question_file, files=files_for_choices)
104         if not question:
105             return self.error_response("Question not found", status_code=404)
106
107         return self.success_response(
108             data=question.to_dict(),
109             message="Question updated successfully"
110         )
111     except BadRequest as e:
112         return self.error_response(str(e))
113     except Exception as e:
114         print(f"Error updating question: {e}")
115         return self.error_response("Failed to update question", status_code=500)
```

4.2.2.3.12 Quiz Controller

```
app > controllers > api > quiz_controller.py
48 # --- CRUD operations for Quizzes ---
49 def get_all_quizzes(self) -> Tuple[Dict[str, Any], int]:
50     """Get all quizzes."""
51     quizzes = self.service.get_all_quizzes()
52     return self.success_response(data=[quiz.to_dict() for quiz in quizzes])
53
54 def get_quiz(self, quiz_id: int) -> Tuple[Dict[str, Any], int]:
55     """Get a specific quiz by its ID."""
56     quiz = self.service.get_quiz_by_id(quiz_id)
57     if not quiz:
58         return self.error_response("Quiz not found", status_code=404)
59     return self.success_response(data=quiz.to_dict())
60
61 def get_quiz_by_level_id(self, level_id: int) -> Tuple[Dict[str, Any], int]:
62     """Get a quiz associated with a specific level ID."""
63     quiz = self.service.get_quiz_by_level_id(level_id)
64     if not quiz:
65         return self.error_response("Quiz not found for this level ID", status_code=404)
66     return self.success_response(data=quiz.to_dict())
```

4.2.2.3.13 Base Controller

```
app > controllers > api > base_controller.py
10 class BaseController:
11
12     def __init__(self, name: str, import_name: str, url_prefix: Optional[str] = None):
13
14         self.blueprint = Blueprint(name, import_name, url_prefix=url_prefix)
15         self._register_error_handlers()
16
17     def _register_error_handlers(self) -> None:
18         """Register common error handlers for the blueprint."""
19         @self.blueprint.errorhandler(HTTPException)
20         def handle_http_error(error: HTTPException) -> Tuple[Dict[str, Any], int]:
21             """Handle HTTP exceptions."""
22             response = {
23                 'error': error.name,
24                 'message': error.description,
25                 'status_code': error.code
26             }
27             return jsonify(response), error.code
```

4.2.3 AI & Deep-learning

In the **LinguaZone** project, Artificial Intelligence (AI), specifically Deep Learning (DL), plays a foundational and transformative role. It is not merely an add-on feature but the core engine that drives the application's unique value proposition: intelligent, real-time grammatical feedback from spoken language. The implementation of AI and Deep Learning is central to creating an interactive and effective learning experience that surpasses traditional language learning methods.

Deep Learning, a subfield of machine learning, utilizes artificial neural networks with multiple layers (deep neural networks) to learn complex patterns from large amounts of data. This capability makes it exceptionally well-suited for tasks involving raw, unstructured data like audio and text, which are fundamental to language.

4.2.3.1 Speech Recognition : DeepSpeech

An open-source, end-to-end deep learning model that offers efficient and accurate speech-to-text conversion.

DeepSpeech provides on-device processing capabilities, which enhances privacy and reduces dependency on external APIs, making it a cost-effective solution.

Additionally, it is lightweight and can run efficiently on resource-constrained devices, ensuring better performance and scalability.

Given the importance of real-time feedback in language learning, DeepSpeech's ability to process audio quickly and integrate seamlessly with our backend (Flask) makes it a suitable choice.

Furthermore, having full control over the model allows for potential fine-tuning and customization to improve accuracy for specific language learning scenarios."

4.2.3.1.1 Environment Setup

The environment setup was carried out using Python 3.8. The following steps were performed to set up the Python environment and install required packages:

1. **Create a Python 3.8 virtual environment**
2. **Install required packages**
3. **Install DeepSpeech**

4.2.3.1.1.1 Required Dependencies

- Python 3.8
 - numpy==1.19.5
 - tensorflow==2.5.0
 - deepspeech==0.9.3
 - pyaudio
 - keyboard
-

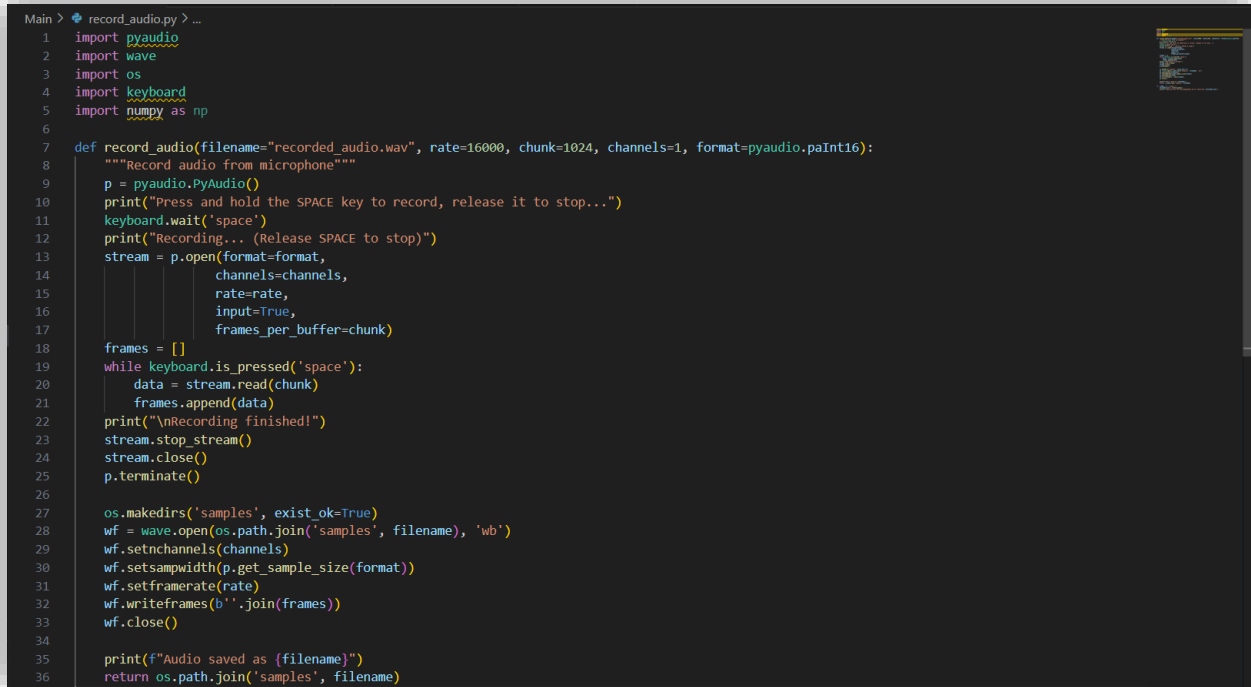
4.2.3.1.1.2 Model Files

- **Model File:** deepspeech-0.9.3-models.pbmm (189 MB)
 - **Scorer File:** deepspeech-0.9.3-models.scorer (953 MB)
-

4.2.3.1.2 Implementation

4.2.3.1.2.1 Audio Recording Implementation

A Python script `record_audio.py` was created to record audio.



```
Main > record_audio.py > ...
1 import pyaudio
2 import wave
3 import os
4 import keyboard
5 import numpy as np
6
7 def record_audio(filename="recorded_audio.wav", rate=16000, chunk=1024, channels=1, format=pyaudio.paInt16):
8     """Record audio from microphone"""
9     p = pyaudio.PyAudio()
10    print("Press and hold the SPACE key to record, release it to stop...")
11    keyboard.wait('space')
12    print("Recording... (Release SPACE to stop)")
13    stream = p.open(format=format,
14                   channels=channels,
15                   rate=rate,
16                   input=True,
17                   frames_per_buffer=chunk)
18    frames = []
19    while keyboard.is_pressed('space'):
20        data = stream.read(chunk)
21        frames.append(data)
22    print("\nRecording finished!")
23    stream.stop_stream()
24    stream.close()
25    p.terminate()
26
27    os.makedirs('samples', exist_ok=True)
28    wf = wave.open(os.path.join('samples', filename), 'wb')
29    wf.setnchannels(channels)
30    wf.setsampwidth(p.get_sample_size(format))
31    wf.setframerate(rate)
32    wf.writeframes(b''.join(frames))
33    wf.close()
34
35    print(f"Audio saved as {filename}")
36    return os.path.join('samples', filename)
```

The functionality includes:

- **Recording audio using PyAudio.**
- **Saving the recording as a WAV file** with the following specifications:
 - 16kHz sample rate
 - Mono channel
 - 16-bit sample width
- **Space-bar controlled recording** for start/stop functionality.
- **Saving the recording as 'recorded_audio.wav'.**

4.2.3.1.2.2 Speech Recognition Implementation

The script `test_deepspeech.py` was modified to perform the following:

- **Load the DeepSpeech model and scorer.**
- **Process WAV files** for speech-to-text conversion.
- **Display transcription results.**

```
Main > test_deepspeech.py > ...
1  import deepspeech
2  import wave
3  import numpy as np
4  import os
5
6  def read_wav_file(filename):
7      with wave.open(filename, 'rb') as w:
8          rate = w.getframerate()
9          frames = w.getnframes()
10         buffer = w.readframes(frames)
11         return buffer, rate
12
13 current_dir = os.path.dirname(os.path.abspath(__file__))
14 parent_dir = os.path.dirname(current_dir)
15 model_file_path = 'D:/Graduation Project (1)/AI Development Code/models/deepspeech-0.9.3-models.pbmm'
16 scorer_file_path = 'D:/Graduation Project (1)/AI Development Code/models/deepspeech-0.9.3-models.scorer'
17 audio_file_path = 'D:/Graduation Project (1)/AI Development Code/samples/recorded_audio.wav'
18
19 print("Loading model...")
20 try:
21     model = deepspeech.Model(model_file_path)
22     model.enableExternalScorer(scorer_file_path)
23     print(f"Processing audio file: {audio_file_path}")
24
25     buffer, rate = read_wav_file(audio_file_path)
26     data16 = np.frombuffer(buffer, dtype=np.int16)
27
28     text = model.stt(data16)
29
30     print('-' * 50)
31     print('Transcription:', text)
32     print('-' * 50)
33 except Exception as e:
34     print(f"Error: {str(e)}")
35     print(f"Model path: {model_file_path}")
36     print(f"Scorer path: {scorer_file_path}")
37     print(f"Audio path: {audio_file_path}")
```

4.2.3.1.2.3 Audio Verification

A script `check_audio.py` was created to verify the following audio properties:

- **Sample rate** (16000 Hz).
- **Number of channels** (Mono).
- **Sample width** (16-bit).
- **Duration and other parameters.**

```
samples > check_audio.py > ...
1  import wave
2  import os
3
4  def check_audio_file(filename):
5      with wave.open(filename, 'rb') as wav:
6          print(f"Number of channels: {wav.getnchannels()}")
7          print(f"Sample width: {wav.getsampwidth()} bytes")
8          print(f"Frame rate (sampling frequency): {wav.getframerate()} Hz")
9          print(f"Number of frames: {wav.getnframes()}")
10         print(f"Parameters: {wav.getparams()}")
11         duration = wav.getnframes() / wav.getframerate()
12         print(f"Duration: {duration:.2f} seconds")
13
14     audio_path = os.path.join('samples', 'recorded_audio.wav')
15     print(f"Checking audio file: {audio_path}")
16     check_audio_file(audio_path)
```

4.2.3.1.3 Testing

Tested using a recording sample script `record_audio.py` and using **DeepSpeech** to transcribe the recorded audio file by `test_deepspeech.py` ... **the model worked correctly but needs more accuracy.**

4.2.3.1.4 Training

To enhance the accuracy of pronunciation evaluation, the pre-trained **DeepSpeech v0.9.3** model was further trained using a large-scale dataset to improve its performance.

the model was fine-tuned using **Speech Commands v0.02**, a dataset containing high quality labeled speech recordings and consists of thousands of short spoken words and phrases, allowing the model to learn varied pronunciation patterns.

4.2.3.1.4.1 Overview of the Dataset

The Speech Commands Dataset v0.02 is a collection of one-second audio files, each containing a single spoken word. It is primarily designed for training and evaluating machine learning models to recognize a small set of spoken commands.

- **Size:** Approximately 105,000 audio files.
- **Duration:** Each audio file is exactly one second long.
- **Format:** WAV files (16-bit, 16kHz mono).
- **Contributors:** Recorded by thousands of different people, contributing to voice diversity.

4.2.3.1.4.1.1 Core Command Words (30 Classes)

The dataset primarily contains 30 distinct spoken words, each in its own subdirectory. These are the target labels for most classification models:

- bed, bird, cat, dog, down, eight, five, four, go, happy, house, left, marvin, nine, no, off, on, one, right, seven, six, three, two, up, yes, zero.

4.2.3.1.4.1.2 Data Splits (Implicit)

While the v0.02 dataset doesn't explicitly come with predefined training, validation, and test sets in separate folders, it includes `testing_list.txt` and `validation_list.txt` files. These files list the relative paths to the audio files designated for the testing and validation sets, respectively. All other files are typically used for training.

- `testing_list.txt`: Paths to files intended for the test set.
- `validation_list.txt`: Paths to files intended for the validation set.
- **Training Set**: All files not listed in `testing_list.txt` or `validation_list.txt`.

This implicit splitting mechanism ensures that models are evaluated on unseen data, preventing overfitting and providing a more accurate measure of generalization performance.

4.2.3.1.4.2 GPU Initialization

To optimize the training process and improve performance, GPU acceleration was enabled by utilizing **CUDA 10.0.130** and **cuDNN 7.4.1.5**, which significantly enhanced the training speed and efficiency.

```
... PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
❖ (bart-venv) PS D:\Graduation Project (1)\AI Development Code\BART> python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] o
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print("CUDA available:", torch.cuda.is_available())
CUDA available: True
>>> print("Device name:", torch.cuda.get_device_name(0))
Device name: NVIDIA GeForce RTX 3050 6GB Laptop GPU
Indexing completed.
```

The training process involved multiple iterations, optimizing the model's ability to distinguish phonemes and evaluate pronunciation accuracy.

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
s experimental feature.
state_dict = torch.load(resolved_archive_file, map_location="cpu")
Epoch 1/3: 100%|████████████████████| 249527/249527 [26:36:46<00:00, 2.60it/s, loss=0.0187]
Epoch 1/3 completed. Average Loss: 0.0378
Epoch 2/3: 100%|████████████████████| 249527/249527 [26:40:31<00:00, 2.60it/s, loss=0.00363]
Epoch 2/3 completed. Average Loss: 0.0292
Epoch 3/3: 100%|████████████████████| 249527/249527 [26:43:47<00:00, 2.59it/s, loss=0.014]
Epoch 3/3 completed. Average Loss: 0.0283
```

4.2.3.2 Grammar Correction : BART

BART is an open-source, end-to-end deep learning model designed for a variety of natural language processing tasks, including text generation, translation, and grammar correction.

BART was utilized for grammar correction tasks, where its ability to process text efficiently and provide accurate corrections plays a pivotal role in improving language quality.

Given the importance of real-time grammar feedback in language learning, BART's efficiency in generating corrected text quickly and its seamless integration with our backend (Flask) made it an ideal choice for the task. The model's architecture allows it to handle complex language structures, making it effective for grammar correction and enhancing the overall user experience.

Furthermore, the ability to fine-tune BART for specific correction tasks, such as grammar mistakes, enables customization to improve accuracy for particular language learning contexts.

4.2.3.2.1 Environment Setup

The environment setup for BART was carried out using Python 3.8. The following steps were performed to set up the Python environment and install the required packages:

- 1. Create a Python 3.8 Virtual Environment**
 - 2. Install Required Packages**
 - 3. Install BART and Required Libraries**
-

4.2.3.2.1.1 Required Dependencies

- Python 3.8
 - numpy==1.21.0
 - torch==1.9.0
 - transformers==4.9.0
 - sentencepiece==0.1.96
-

4.2.3.2.1.2 Model Files

- **Model File:** facebook/bart-large (Pre-trained model weights)
 - **Tokenizer File:** facebook/bart-large-tokenizer (Tokenizer used to process input text)
-

4.2.3.2.1 Implementation Steps

A script `grammar_correction.py` was created to perform the following tasks:

- Load the pre-trained BART model and its tokenizer.
- Process input text and apply the model to generate grammatically corrected text.
- Output the corrected text for review.

```
BART > grammar_correction.py > ...
1  # Import the necessary libraries
2  from transformers import BartForConditionalGeneration, BartTokenizer
3
4  # Load the pre-trained BART model and tokenizer
5  model_name = 'facebook/bart-large'
6  model = BartForConditionalGeneration.from_pretrained("d:/Graduation Project (1)/AI Development Code/BART/model/fine_tuned_bart")
7  tokenizer = BartTokenizer.from_pretrained("d:/Graduation Project (1)/AI Development Code/BART/model/fine_tuned_bart")
8  # Define a function to correct grammar using the BART model
9  # Tokenize the input text
10 def correct_grammar(text):
11     inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
12     # Generate the corrected output
13     summary_ids = model.generate(
14         inputs["input_ids"],
15         num_beams=4,
16         max_length=128,
17         no_repeat_ngram_size=3
18     )
19     # Decode and return the corrected text
20     corrected_text = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
21     return corrected_text.strip()
22
23 # Input text to be corrected
24 input_text = "He am a software engineer." # Replace this with any sentence you need to correct
25
26 # Get the corrected text
27 corrected_text = correct_grammar(input_text)
28 print("Corrected Text:", corrected_text)
```

4.2.3.2.3 Testing

The model was tested using sample text data to verify its accuracy in grammar correction. After running the `grammar_correction.py` script, the model correctly corrected a range of grammatical errors, but some edge cases were still identified, suggesting the need for additional fine-tuning.

4.2.3.2.4 Training

To improve the accuracy of grammar corrections, the pre-trained BART model was further fine-tuned using the Lang-8 Corpus dataset. This dataset, containing sentences with common grammatical errors, enabled the model to learn patterns in grammar mistakes and their corrections.

4.2.3.2.4.1 Overview of the Lang-8 Corpus

The original **Lang-8 corpus** is derived from the Lang-8 website, a language exchange platform where native speakers correct texts written by language learners. This platform provides a rich source of genuine learner errors and their corresponding corrections by human experts.

- **Source:** Text from the Lang-8 online language exchange platform.
- **Content:** Pairs of original learner sentences (with errors) and their human-corrected versions.
- **Significance:** It contains authentic errors made by non-native speakers, reflecting the challenges faced in real-world language acquisition.

4.2.3.2.4.1.1 The .m2 Format (Maximally Informative M2)

The .m2 format is a standard annotation format designed for grammatical error correction tasks. It allows for the precise representation of edits (additions, deletions, substitutions) between an original sentence and its corrected version. The `auto` in `train.auto.bea19.m2` indicates that these annotations were automatically generated or aligned, likely from the original Lang-8 corrections.

4.2.3.2.4.1.2 Data Split

The dataset was split into training, validation, and test sets to ensure that the model was evaluated thoroughly throughout the fine-tuning process. This allowed for regular performance checks, ensuring that the model could generalize well to unseen data and provide high-quality grammar corrections.

4.2.3.2.4.2 GPU Initialization

The training process was optimized by enabling GPU acceleration, which significantly sped up the fine-tuning process. Using CUDA 12.8.1 and cuDNN 9.8.0, the model's training was enhanced, reducing the time required for each epoch and enabling faster iteration.



```
... PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
❖ (bart-venv) PS D:\Graduation Project (1)\AI Development Code\BART> python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] o
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print("CUDA available:", torch.cuda.is_available())
CUDA available: True
>>> print("Device name:", torch.cuda.get_device_name(0))
Device name: NVIDIA GeForce RTX 3050 6GB Laptop GPU
Indexing completed.
```

During training, multiple iterations were carried out to fine-tune the model's understanding of grammar and improve its accuracy in generating corrections.

4.2.3.3 Final Integration : DeepSpeech + BART

This system integrates two models—**DeepSpeech** for speech-to-text transcription and **BART** for grammar correction—into a single web application endpoint. The endpoint receives audio files, transcribes the audio into text using the DeepSpeech model, and then corrects any grammatical errors in the transcribed text using the BART model. This entire process is executed on the local server without the need for external APIs, providing a seamless and efficient workflow.

4.2.3.3.1 Architecture

The application leverages **Flask**, a lightweight web framework, to create a REST API endpoint. This endpoint, `/process_audio`, accepts POST requests with audio files attached. The audio file undergoes the following steps:

1. **Audio File Processing:**

- The uploaded audio file is saved to the server.
- It is then passed to the **DeepSpeech model** to transcribe the audio into text.

2. **Text Processing:**

- The transcribed text is then passed to the **BART model** for grammar correction.

3. **Return Corrected Text:**

- The corrected text is returned to the client as a JSON response.

4.2.3.3.2 Integration of DeepSpeech and BART Models

4.2.3.3.2.1 DeepSpeech Model for Audio Transcription

- **Purpose:** DeepSpeech is used to convert speech (audio) into text.
- **Workflow:** The audio file is processed through the following steps:
 - The audio is read from a `.wav` file.
 - The audio data is converted into a numpy array.
 - The numpy array is fed into the **DeepSpeech model** for transcription.
 - The transcription (text) is returned to be processed by BART for grammar correction.

4.2.3.3.2.2 BART Model for Grammar Correction

- **Purpose:** BART is used to perform grammar correction on the transcribed text from DeepSpeech.
- **Workflow:** Once the text is transcribed, it undergoes the following process:
 - The transcribed text is tokenized using the **BART tokenizer**.
 - The tokenized input is passed through the **fine-tuned BART model** to generate a grammatically corrected version of the text.
 - The corrected text is then decoded and returned to the user.

4.2.3.3.2.3 Combining the Two Models into a Single Endpoint

- The integration of these two models is achieved by sequentially invoking both models within a single Flask endpoint.
 - **Step 1:** The server receives the audio file via a POST request at `/process_audio`.
 - **Step 2:** The audio file is saved on the server and passed to the `transcribe_audio` function, which uses the **DeepSpeech model** to convert speech to text.
 - **Step 3:** The transcribed text is then passed to the `correct_grammar` function, which uses the **BART model** to correct any grammatical errors in the text.
 - **Step 4:** The corrected text is returned to the client as a JSON response.

4.2.3.3.3 Endpoints Workflow

4.2.3.3.3.1 /transcribe_audio Endpoint Workflow

This endpoint is designed to provide immediate transcription feedback to the user, allowing them to verify what they said before any grammar correction is applied.

```

DeepSpeech-BART > Scripts > server.py > transcribe_audio_endpoint
36 @app.route('/transcribe_audio', methods=['POST']) # Endpoint for DeepSpeech transcription
37 def transcribe_audio_endpoint():
38     if 'audio' not in request.files:
39         return jsonify({"error": "No audio file found"}), 400
40
41     audio_file = request.files['audio']
42     audio_path = os.path.join('uploads', audio_file.filename)
43
44     # Ensure 'uploads' directory exists
45     os.makedirs('uploads', exist_ok=True)
46
47     # Save the audio file to the server
48     audio_file.save(audio_path)
49
50     # Transcribe audio using DeepSpeech
51     try:
52         transcribed_text = transcribe_audio(audio_path)
53         return jsonify({"transcribed_text": transcribed_text})
54     except Exception as e:
55         return jsonify({"error": str(e)}), 500

```

The /transcribe_audio endpoint processes the audio file in the following sequence:

- 1. Client Uploads Audio:**

- The client (e.g., Mobile/Web App) sends a POST request with an audio file attached (e.g., .wav file) to the /transcribe_audio endpoint.

- 2. Audio File Handling:**

- The server receives the audio file and temporarily saves it (e.g., to a temp directory or processes directly from stream) on the local machine.

- 3. Speech-to-Text Transcription:**

- The saved audio file is passed to the DeepSpeech model via the transcribe_audio function.
- DeepSpeech processes the audio and returns the raw transcribed text.

- 4. Response to Client:**

- The transcribed text is returned to the client as a JSON object containing the key transcribed_text. This text is then displayed in the application for the user to review.

4.2.3.3.2 /process_audio Endpoint Workflow

```
DeepSpeech-BART > Scripts > server.py > transcribe_audio_endpoint
59 @app.route('/process_audio', methods=['POST']) # Endpoint for full processing (transcription + grammar correction)
60 def process_audio():
61     if 'audio' not in request.files:
62         return jsonify({"error": "No audio file found"}), 400
63
64     audio_file = request.files['audio']
65     audio_path = os.path.join('uploads', audio_file.filename)
66
67     # Ensure 'uploads' directory exists
68     os.makedirs('uploads', exist_ok=True)
69
70     # Save the audio file to the server
71     audio_file.save(audio_path)
72
73     # Step 1: Transcribe audio using DeepSpeech
74     transcribed_text = transcribe_audio(audio_path)
75
76     # Step 2: Correct grammar using BART
77     corrected_text = correct_grammar(transcribed_text)
78
79     return jsonify({"corrected_text": corrected_text})
```

processes the audio file in the following sequence:

1. Client Uploads Audio:

- The client sends a POST request with an audio file attached (e.g., .wav file) to the /process_audio endpoint.

2. Audio File Handling:

- The server receives the audio file and saves it to the uploads directory on the local machine.

3. Speech-to-Text:

- The saved audio file is passed to the **DeepSpeech** model via the transcribe_audio function.
- DeepSpeech transcribes the audio into text.

4. Grammar Correction:

- The transcribed text is passed to the **BART** model via the correct_grammar function.
- BART processes the text and returns a grammatically corrected version.

5. Response to Client:

- The corrected text is returned to the client as a JSON object containing the key corrected_text.

4.2.3.3.4 Model Files

1. DeepSpeech:

- **Model File:** deepspeech-0.9.3-models.pbmm (The main model used for speech-to-text transcription).
- **Scorer File:** deepspeech-0.9.3-models.scorer (Used to improve transcription accuracy).

2. BART:

- **Fine-Tuned Model:** The fine_tuned_bart directory containing the trained BART model and its tokenizer for grammar correction.

4.2.3.3.5 System Requirements

- **Python 3.8+:** The system requires Python version 3.8 or higher.
- **DeepSpeech:** For speech-to-text transcription.
- **Transformers:** For utilizing the BART model and tokenizer.
- **Flask:** For setting up the web application and exposing the endpoint.
- **Other Libraries:**
 - **numpy:** Used for handling audio data.
 - **wave:** Used for reading .wav audio files

4.2.4 Data Layer (MySQL Database)

The Data Layer of the LinguaZone system is built upon a robust MySQL relational database, designed to store and manage all persistent application data. This layer is critical for maintaining data integrity, ensuring efficient data retrieval, and supporting the application's core functionalities.

4.2.4.1 Database Schema

The database schema implements the Entity-Relationship Diagram (ERD) designed for LinguaZone. It comprises a set of interconnected tables that store information about users, learning content (levels, sections, quizzes, questions, question choices), and the relationships between them. This structured schema ensures data integrity through defined primary and foreign keys, enforces relationships, and facilitates efficient querying for various application needs. The primary tables include:

- **users:** Stores user authentication and profile information.
- **levels:** Defines different language proficiency levels.
- **sections:** Organizes learning content within specific levels.
- **quizzes:** Manages quiz definitions associated with levels.
- **questions:** Stores individual learning questions of various types.
- **quiz_questions:** A junction table linking quizzes to questions (many-to-many relationship).
- Additional tables for UserProgress, GrammarCorrections, and AudioTranscriptions (as outlined in the ERD documentation).

4.2.4.2 Data Access Layer (via SQLAlchemy)

All interactions with the MySQL database are managed through a Data Access Layer implemented using **SQLAlchemy**. SQLAlchemy serves as an Object-Relational Mapper (ORM), providing a powerful and consistent Pythonic interface for the Backend API modules to store, retrieve, update, and delete data. This abstraction eliminates the need for writing raw SQL queries directly in the application logic, enhancing development speed, reducing the risk of SQL injection vulnerabilities, and improving code readability and maintainability.

SQLAlchemy allows developers to work with database tables as Python objects (models), mapping database rows to instances of these models. This object-oriented approach simplifies complex database operations and relationships.

4.2.4.3 Database Models (/models folder)

Within the /models folder of the backend application, specific Python classes are defined using SQLAlchemy's declarative base to represent each database table. These models not only define the table structure (columns, data types, primary keys, foreign keys) but also establish the relationships between entities, making data manipulation intuitive and object-oriented.

Here's a breakdown of the key models:

4.2.4.3.1 User Model (user.py)

```
app > models > user.py ...
6 class UserRole(enum.Enum):
7     USER = "user"
8     ADMIN = "admin"
9
10 class User(db.Model):
11     __tablename__ = 'users'
12
13     id = db.Column(db.Integer, primary_key=True)
14     email = db.Column(db.String(120), unique=True, nullable=False)
15     username = db.Column(db.String(80), unique=True, nullable=False)
16     password_hash = db.Column(db.String(255))
17     role = db.Column(db.Enum(UserRole), nullable=False, default=UserRole.USER)
18     is_verified = db.Column(db.Boolean, default=False)
19     verification_code = db.Column(db.String(6))
20     verification_code_expires = db.Column(db.DateTime)
21     created_at = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
22     updated_at = db.Column(db.DateTime, nullable=False, default=datetime.utcnow, onupdate=datetime.utcnow)
23
24     def set_password(self, password):
25         self.password_hash = generate_password_hash(password)
26
27     def check_password(self, password):
28         return check_password_hash(self.password_hash, password)
29
30     def to_dict(self):
31         return {
32             'id': self.id,
33             'email': self.email,
34             'username': self.username,
35             'role': self.role.value,
36             'is_verified': self.is_verified,
37             'created_at': self.created_at.isoformat(),
38             'updated_at': self.updated_at.isoformat(),
39         }
40
```

- **Purpose:** Represents the `users` table, storing all user-related information.

4.2.4.3.2 Level Model (level.py)

```
app > models > level.py
1 from app import db
2 from datetime import datetime
3
4 class Level(db.Model):
5     __tablename__ = 'levels'
6
7     id = db.Column(db.Integer, primary_key=True)
8     name = db.Column(db.String(100), nullable=False, unique=True)
9     description = db.Column(db.Text)
10    image_url = db.Column(db.String(255))
11    created_at = db.Column(db.DateTime, default=datetime.utcnow)
12    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
13
14    # Relationships
15    sections = db.relationship('Section', back_populates='level', lazy=True, cascade='all, delete-orphan')
16
17    def to_dict(self):
18        return {
19            'id': self.id,
20            'name': self.name,
21            'description': self.description,
22            'image_url': self.image_url,
23            'created_at': self.created_at.isoformat() if self.created_at else None,
24            'updated_at': self.updated_at.isoformat() if self.updated_at else None
25        }
26
27    def __repr__(self):
28        return f'<level {self.name}>'

```

- **Purpose:** Represents the `levels` table, organizing language proficiency levels.

4.2.4.3.3 Section Model (section.py)

```
app > models > section.py > --
1 from app import db
2 from datetime import datetime
3
4 class Section(db.Model):
5     __tablename__ = 'sections'
6
7     id = db.Column(db.Integer, primary_key=True)
8     name = db.Column(db.String(100), unique=True, nullable=False)
9     description = db.Column(db.Text, nullable=True)
10    image = db.Column(db.String(255), nullable=True)
11    level_id = db.Column(db.Integer, db.ForeignKey('levels.id'), nullable=False)
12    created_at = db.Column(db.DateTime, default=datetime.utcnow)
13    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
14
15    # Relationships
16    level = db.relationship('level', back_populates='sections')
17    questions = db.relationship('Question', back_populates='section', lazy=True, cascade='all, delete-orphan')
18
19    def to_dict(self):
20        return {
21            'id': self.id,
22            'name': self.name,
23            'description': self.description,
24            'image': self.image,
25            'level_id': self.level_id,
26            'created_at': self.created_at.isoformat(),
27            'updated_at': self.updated_at.isoformat()
28        }
```

- **Purpose:** Represents the `sections` table, defining subsections within each learning level.

4.2.4.3.4 Question Model (question.py)

```
app > models > question.py > Question > get_correct_answers
8 class Question(db.Model):
9     __tablename__ = 'questions'
10
11    id = db.Column(db.Integer, primary_key=True)
12    section_id = db.Column(db.Integer, db.ForeignKey('sections.id'), nullable=False)
13    section = db.relationship('Section', back_populates='questions')
14    question_type = db.Column(db.Enum(QuestionType), nullable=False)
15    question_content = db.Column(db.String(255), nullable=False)
16    answer_type = db.Column(db.Enum(AnswerType), nullable=False)
17    correct_answer = db.Column(db.String(255), nullable=True)
18    created_at = db.Column(db.DateTime, default=datetime.utcnow)
19    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
20
21    # Relationships
22    choices = db.relationship('QuestionChoice', back_populates='question', cascade='all, delete-orphan')
23
24    def to_dict(self):
25        return {
26            'id': self.id,
27            'section_id': self.section_id,
28            'question_type': self.question_type.value,
29            'question_content': self.question_content,
30            'answer_type': self.answer_type.value,
31            'correct_answer': self.correct_answer,
32            'choices': [choice.to_dict() for choice in self.choices],
33            'created_at': self.created_at.isoformat(),
34            'updated_at': self.updated_at.isoformat()
35        }
```

- **Purpose:** Represents the `questions` table, storing individual questions used in sections and quizzes.

4.2.4.3.5 Quiz Model (quiz.py)

```

app > models > quiz.py > QuizChoice > to_dict
21 class Quiz(db.Model):
22     """
23     Represents a quiz, which is associated with a specific level.
24     Each level can have one quiz.
25     """
26     __tablename__ = 'quizzes'
27
28     id = db.Column(db.Integer, primary_key=True)
29
30     # Foreign key to the Level model. Each quiz is tied to a level.
31     level_id = db.Column(db.Integer, db.ForeignKey('levels.id'), nullable=False, unique=True) # Unique per level
32     name = db.Column(db.String(100), nullable=False)
33     description = db.Column(db.Text, nullable=True)
34     created_at = db.Column(db.DateTime, default=datetime.utcnow)
35     updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
36
37     # Relationships
38     level = db.relationship('Level', backref='quiz', uselist=False) # One-to-one relationship with Level
39
40     # One-to-many relationship with QuizQuestion.
41     # 'cascade' ensures that when a Quiz is deleted, its associated QuizQuestions are also deleted.
42     # 'order_by' ensures questions are always retrieved in order.
43     quiz_questions = db.relationship(
44         'QuizQuestion',
45         back_populates='quiz',
46         cascade='all, delete-orphan',
47         order_by='QuizQuestion.order_in_quiz'
48     )

```

- **Purpose:** Represents the `quizzes` table, defining assessment modules within the application.

These SQLAlchemy models, defined in their respective files within the `/models` folder, form the backbone of the LinguaZone application's data management, enabling seamless interaction between the Python backend and the MySQL database.

4.3 Challenges faced and how they were resolved.

4.3.1 Maintaining Timer State After Page Reload

Challenge:

Initially, the countdown timer would reset every time the page was refreshed, which broke the continuity of the activity.

Solution:

Used localStorage to store the remaining time. On page load, the script checks for any saved time and resumes the countdown accordingly.

4.3.2 Managing Button Active State Dynamically

Challenge:

Making sure only one confidence button (Low, Medium, High) stays active at a time, and that the visual state reflects user selection.

Solution:

Used JavaScript to loop over all buttons, remove any previously applied .active class, then apply it to the clicked button.

4.3.3 Preventing Invalid User Interaction

Challenge:

Preventing users from proceeding before completing all matches or giving feedback.

Solution:

JavaScript checks were added before proceeding to the result page. If the required conditions weren't met, a message appears and the user is blocked from continuing.

4.3.4 Responsive Website Design for Different Screens

Challenge:

Making the layout visually consistent and usable on both large and small screens, especially since the interface includes images, buttons, and background shapes.

Solution:

Used flexible units in CSS (like %, vh, vw), and styled each section carefully using media-friendly layout techniques like flexbox and relative positioning.

4.3.5 UI Responsiveness for mobile devices

Challenge: Managing different screen sizes and keeping the UI responsive was also a focus area.

Solution: We used Media Query and flexible widgets like Expanded and Flexible to ensure that the layout adapts well to all device sizes.

Chapter 5

Testing & Evaluation

5.1 Testing Strategies

A multi-faceted testing approach was implemented throughout the development lifecycle, combining automated and manual methods to cover different layers of the application and ensure comprehensive quality assurance.

5.1.1 Unit Testing

- **Purpose:** To verify the correctness of individual components or functions in isolation. This ensures that each small piece of code behaves as expected before integration.
- **Application in LinguaZone:**
 - **Backend (Python/Flask):** Unit tests were written for individual API endpoints, utility functions (e.g., password hashing, JWT token generation), database ORM operations (e.g., creating a user, fetching a lesson), and specific logic within AI integration modules. Libraries like `pytest` were used, often with mocking frameworks to isolate dependencies (e.g., mocking database calls or external AI service responses).
 - **Frontend (Flutter/Dart):** Unit tests focused on the logic of individual widgets, state management providers/blocs, and utility classes. This ensured that UI components rendered correctly based on state and that business logic within the frontend was sound. `flutter_test` framework was utilized for this.

- **AI Services (DeepSpeech/BART):** Unit tests verified the correct loading of models, basic inference calls, and the format of the output (e.g., ensuring DeepSpeech returns a string, BART returns a corrected string). These tests did not necessarily evaluate accuracy but ensured the core functionality of the model wrapper was sound.

5.1.2 Integration Testing

- **Purpose:** To verify that different modules or services of the system work together correctly when integrated. This catches issues arising from interfaces or data flow between components.
- **Application in LinguaZone:**
 - **Frontend-Backend Integration:** Tests simulated user interactions (e.g., login, submitting a quiz, recording audio) and verified that the frontend correctly sent requests to the backend and processed the responses. This involved testing API contract adherence.
 - **Backend-Database Integration:** Tests ensured that the Backend API correctly interacted with the MySQL database through SQLAlchemy, verifying data persistence, retrieval, and relationship management across multiple entities (e.g., creating a level, adding sections and questions to it, then fetching them).
 - **Backend-AI Service Integration:** Critical tests focused on the complete flow involving AI. For instance, sending a simulated audio file to the `/process_audio` endpoint and verifying that the entire chain (Backend -> DeepSpeech -> BART -> Backend) executed correctly and returned a valid, corrected text. This validated the inter-service communication and data handoff.

- **Container Integration:** Tests were performed to ensure that all Dockerized services (Flask API, DeepSpeech service, BART service) could communicate effectively within their containerized environment.

5.1.3 User Acceptance Testing (UAT) / System Testing

- **Purpose:** To verify that the entire system meets the specified requirements and user expectations in a real-world scenario. This often involves real end-users.
- **Application in LinguaZone:**
 - **Scenario-Based Testing:** Test cases were derived directly from defined user stories and use cases (e.g., "As a learner, I can sign up and verify my email," "As a learner, I can complete a speech-based exercise and receive grammar feedback").
 - **Usability Testing:** A group of target users (language learners) was involved in testing sessions. Their interactions were observed, and feedback was collected on the user interface, overall flow, clarity of instructions, and the helpfulness of the grammar feedback.
 - **Cross-Platform Validation:** The complete application was tested extensively on various mobile devices (different Android versions, iOS devices) and web browsers to ensure consistent functionality and appearance.
 - **Accessibility Testing:** Basic checks were performed to ensure the application was usable for individuals with common disabilities (e.g., keyboard navigation, sufficient color contrast).

- **Performance under Load (Basic):** Limited load testing was performed to identify obvious bottlenecks under a moderate number of concurrent users.

5.2 Performance Metrics

To quantitatively assess the effectiveness and efficiency of **LinguaZone**, several key performance metrics were monitored and evaluated across different layers of the system.

5.2.1 AI Model Performance Metrics

These metrics are crucial for evaluating the core intelligence of the system.

- **Speech-to-Text (DeepSpeech):**
 - **Word Error Rate (WER):** The primary metric for transcription accuracy. It measures the number of errors (substitutions, deletions, insertions) per word in the transcribed output compared to the ground truth. Lower WER indicates higher accuracy.
 - **Character Error Rate (CER):** Similar to WER but calculated at the character level, often used for more fine-grained analysis, especially with non-English languages or specific error types.
 - **Inference Latency:** The time taken for the DeepSpeech model to transcribe a one-second audio clip. Measured from the moment the audio is received by the service to when the transcription is returned.

- **Grammar Error Correction (BART):**

- **M2 Score (F-score on M2 annotations):** The standard metric for GEC tasks, measuring the precision and recall of edits (insertions, deletions, substitutions) compared to human-annotated corrections in the .m2 format. A higher M2 score indicates better error detection and correction.
- **Fluency:** Subjective or semi-automatic evaluation of whether the corrected sentences sound natural and grammatically correct to a human reader, even if the M2 score is high. This can involve human evaluation or metrics like Perplexity.
- **Error Detection Rate:** The percentage of actual grammatical errors in a text that the model successfully identifies.
- **Correction Accuracy:** The percentage of identified errors for which the model provides a correct or acceptable correction.
- **Inference Latency:** The time taken for the BART model to process a given sentence and return the corrected version.

5.2.2 Application Performance Metrics

These metrics assess the overall responsiveness, stability, and scalability of the application.

- **Response Time (Latency):**
 - **API Response Time:** The time taken for the Backend API to respond to various requests (e.g., login, fetch lesson, submit quiz, process audio). Measured in milliseconds.
 - **End-to-End Latency:** The total time from when a user initiates an action (e.g., records speech) to when they receive the final feedback (e.g., corrected text displayed). This includes frontend processing, network travel, backend processing, and AI inference time.
- **Throughput:** The number of requests the system can process per unit of time (e.g., requests per second). This is a key indicator of scalability.
- **Scalability:** The ability of the system to handle an increasing number of users or requests without significant degradation in performance. Assessed by monitoring resource utilization (CPU, memory) under simulated load.
- **Reliability/Availability:** The percentage of time the system is operational and accessible to users. Monitored through uptime metrics and error rates.
- **Resource Utilization:** CPU, memory, and network bandwidth consumption of the frontend, backend, and AI services under various loads.

5.3 Comparison with Existing Solutions

LinguaZone aims to fill specific gaps identified in the literature review, primarily regarding integrated, intelligent grammatical feedback from spoken input. A comparison with leading language learning platforms highlights LinguaZone's unique contributions:

- **Duolingo / LearnMatch:**

- **Strength of Competitors:** Excellent gamification, widespread accessibility, strong for vocabulary building and basic sentence structures. Their speech components often focus on simple word recognition.
- **LinguaZone's Advantage:** While Duolingo might recognize if a word was spoken correctly, it typically lacks deep grammatical analysis of the *entire spoken sentence*. LinguaZone, via DeepSpeech + BART, specifically focuses on transcribing what the user *said* and then providing **detailed grammatical error correction** on that transcribed speech. This moves beyond simple pronunciation checks to actual linguistic accuracy, providing explanations for structural errors.

- **General Grammar Checkers (e.g., Grammarly, LanguageTool):**

- **Strength of Competitors:** Highly effective for written text, offering sophisticated grammar and style suggestions.
- **LinguaZone's Advantage:** These tools are primarily designed for *written* input. LinguaZone integrates this capability directly with *spoken* input, making it a crucial bridge for learners who need to practice speaking and receive grammatical feedback simultaneously.

The sequential processing (speech-to-text, then grammar correction) is a core differentiator that provides comprehensive spoken language practice.

- **Human Tutors:**

- **Strength of Competitors:** Offer highly personalized, nuanced, and context-aware feedback.
- **LinguaZone's Advantage:** While LinguaZone cannot fully replicate human interaction, it provides **scalable, on-demand, and affordable** feedback that human tutors cannot. It democratizes access to sophisticated grammatical analysis, making it available anytime, anywhere, without the cost and scheduling constraints of human instructors.

- **Integration and Cohesiveness:**

- **LinguaZone's Advantage:** Unlike relying on multiple fragmented tools (a speaking app, a separate grammar checker), LinguaZone offers a unified platform where spoken practice directly feeds into intelligent grammar analysis, streamlining the learning process and providing a more cohesive and efficient user experience.

In essence, LinguaZone differentiates itself by providing a focused and intelligent solution for a critical missing piece in automated language learning: **actionable, AI-driven grammatical feedback derived directly from a learner's spoken output**, combined with a user-friendly, cross-platform interface.

Chapter 6

Conclusion & Future Work

6.1 Summary of Contributions

The **LinguaZone** project successfully developed and implemented an innovative AI-powered language learning application that addresses critical gaps in existing solutions, particularly concerning real-time, intelligent grammatical feedback from spoken input. The core contributions of this project are:

- **Integrated AI-Driven Grammar Correction from Speech:** This is the project's most significant contribution. By seamlessly integrating a robust Speech-to-Text (DeepSpeech) model with a powerful Grammar Error Correction (BART) model, LinguaZone provides a unique capability: users can speak freely, and the system not only transcribes their speech but also analyzes it for grammatical accuracy, offering precise corrections and feedback. This moves beyond simple pronunciation checks common in other apps, directly tackling a major challenge for language learners.
- **Comprehensive Language Learning Ecosystem:** Beyond its core AI feature, LinguaZone provides a structured learning environment. The implementation of a well-defined database schema (ERD) and its corresponding SQLAlchemy models (`users`, `levels`, `sections`, `quizzes`, `questions`, etc.) ensures robust content management and user progress tracking. This foundational structure supports various exercise types and a clear progression path.

- **Cross-Platform Accessibility with Unified UX:** The choice and successful implementation using Flutter enabled the delivery of a consistent and high-quality user experience across both web and mobile platforms from a single codebase. This significantly expands the application's reach and accessibility, making intelligent language feedback available to a wider audience without compromising on performance or design integrity.
- **Modular and Scalable Architecture:** The system was designed with a clear separation of concerns, employing a Flask backend API and dedicated, containerized AI microservices. This modular architecture enhances maintainability, simplifies debugging, and provides a scalable foundation that can handle increasing user loads and future feature expansions without significant re-engineering.
- **Addressing Authenticity in Learning:** By leveraging datasets like the Lang-8 corpus (e.g., `lang8.train.auto.bea19.m2`), the BART model was fine-tuned to recognize and correct authentic errors commonly made by non-native speakers, leading to more relevant and practical feedback for learners.

In summary, LinguaZone delivers a unique, integrated, and effective platform that empowers language learners to practice speaking more confidently by providing intelligent, actionable grammatical feedback, thus significantly enhancing their self-correction abilities and accelerating their learning journey.

6.2 Possible Improvements or Extensions for Future Work

The current iteration of **LinguaZone** lays a strong foundation, but several avenues exist for future development and enhancement:

- **Adding Support for Other Languages:** Currently focused on English, a critical and highly demanded extension would be to incorporate support for additional languages (e.g., Spanish, French, German, Arabic, Mandarin). This would involve:
 - Acquiring or fine-tuning language-specific DeepSpeech models for transcription.
 - Training or fine-tuning BART models (or other appropriate GEC models) for grammar correction in the new target languages, requiring relevant error-annotated corpora.
 - Adapting the frontend UI for language switching and potentially localization.
- **Advanced Personalization and Adaptive Learning Paths:**
 - Implement more sophisticated adaptive algorithms that dynamically adjust lesson difficulty and content based on individual user performance, identified weaknesses (e.g., specific grammar rules repeatedly violated), and learning styles.
 - Introduce personalized recommendations for exercises and quizzes.

- **Enhanced Feedback Mechanisms:**

- Provide more granular and interactive feedback, perhaps allowing users to click on corrected phrases to see detailed grammar explanations, rules, or example sentences.
- Introduce phonetic feedback or pronunciation scoring in addition to grammar correction.
- Explore multimodal feedback, such as visual cues alongside text.

- **Diverse Exercise Types and Content Generation:**

- Expand beyond current question types to include more complex tasks like dictation, role-playing simulations, or free-form speech prompts.
- Investigate AI-driven content generation to automatically create new exercises or example sentences based on user proficiency and error patterns.

- **Community and Social Features:**

- Integrate features for learners to interact with each other (e.g., forums, study groups, peer correction).
- Allow users to share their progress or corrected sentences.

- **Gamification and Motivation:**

- Further develop gamification elements beyond basic progress tracking, such as leaderboards, streaks, and more elaborate reward systems to boost user engagement and motivation.

- **Optimizing AI Models for Edge Devices:**

- Explore techniques like model quantization and pruning to create smaller, more efficient versions of DeepSpeech or BART that could potentially run (or partially run) on user devices, reducing latency and backend load.

References

Architecture & Design Literature

- [1] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, Prentice Hall, 2017.
 - [2] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002.
 - [3] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, O'Reilly Media, 2015.
 - [4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
-

Core Technology Documentation

- [5] Flutter.dev, “Flutter Documentation,” 2025. [Online]. Available: <https://flutter.dev/docs>
 - [6] P. Gray, “Flask Documentation,” 2025. [Online]. Available: <https://flask.palletsprojects.com/>
 - [7] Python Software Foundation, “Python 3.x Documentation,” 2025. [Online]. Available: <https://docs.python.org/3/>
 - [8] Oracle, “MySQL Reference Manual,” 2025. [Online]. Available: <https://dev.mysql.com/doc/>
 - [9] Docker Inc., “Docker Documentation,” 2025. [Online]. Available: <https://docs.docker.com/>
-

Data / Framework Sources

[10] Google, “Speech Commands Dataset (v0.02),” 2017.

[11] “DeepSpeech,” Mozilla, 2017–. [Online]. Available:

<https://github.com/mozilla/DeepSpeech>

DeepSpeech’s ASR Method (Listen, Attend and Spell)

[12] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, “Listen, Attend and Spell,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Shanghai, China, Mar. 2016.

Lang-8 Corpus & BEA-2019 Shared Task

[13] K. Mizumoto, H. Yokogiri, and K. Nagata, “Lang-8 Learner Corpora for ESL Error Correction,” *Lang-8*, 2011.

[14] C. Bryant *et al.*, “The BEA-2019 Shared Task on Grammatical Error Correction,” in *Proc. BEA-2019*, Florence, Italy, Jul. 2019.

Foundational Transformer Papers

[15] A. Vaswani *et al.*, “Attention Is All You Need,” in *Proc. 31st Adv. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.

[16] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension,” in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Online, Jul. 2020, pp. 7871–7880, doi: 10.18653/v1/2020.acl-main.703.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv preprint arXiv:1810.04805*, Oct. 2018.

[18] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” in *Proc. 27th Adv. Neural Inf. Process. Syst. (NeurIPS)*, Montréal, Canada, Dec. 2014.