**Listening Activity**

24-25J-103

Final Report

Thisara Dewinda

## B.Sc. (Hons) Degree in Information Technology Specialized in Software Engineering

## Department of Information Technology

## Sri Lanka Institute of Information Technology Sri Lanka

## April 2025

**Listening Activity**

24-25J-103

Final Report

Thisara Dewinda

## B.Sc. (Hons) Degree in Information Technology Specialized in Software Engineering

## Department of Information Technology

## Sri Lanka Institute of Information Technology Sri Lanka

## April 2025

## Declaration

I declare that this is my own work and this report does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

|  |  |  |
| --- | --- | --- |
| Name | Student ID | Signature |
| Dewinda A.G.T | IT21196560 |  |

Signature of the Supervisor Date : 05/04/2025

Signature of the Co-Supervisor Date : 05/04/2024

#### Abstract

The boundaries of technology continue to be eliminated and as a result, language learning and comprehension tools appropriate for children aged 10-12 have emerged. This study attempts to integrate some of the most advanced technologies such as AI-powered listening exercises, adaptive gamification features, personalized education, and predictive analysis to develop a holistic educational platform for learners of this age segment. The AI activities employ natural language processing (NLP) for response assessment and feedback provision, which excel in reinforcing and enhancing language proficiency. At the same time, the adaptive gamification captures students' faces with a facial recognition system, measuring their emotional responses to content in order to adaptively engage students and promote individualized instruction. Furthermore, predictive analytics ensure intervention and content personalization based on the child's monitored progression toward academic goals. This research exemplifies how interactive and adaptive learning environments that motivate students and enhance understanding and academic performance can be achieved by integrating disparate technologies.

Introduction

### Background & Literature Survey

In recent years, language acquisition particularly the development of listening skills has undergone significant transformation due to the influence of modern educational technologies. Traditional language learning models, which often rely on standardized instructional approaches, have not adequately addressed the diverse needs of individual learners. This one-size-fits-all model fails to account for the varying levels of proficiency among students, often resulting in some students falling behind, while others may not be sufficiently challenged. This lack of personalization leads to disengagement and suboptimal learning outcomes, as students do not receive the tailored support they need to thrive.

Listening, being one of the foundational skills in second language acquisition (SLA), plays a crucial role in helping learners develop fluency and comprehension. Traditional language learning methods, grounded in theories like Krashen’s Input Hypothesis, emphasize the importance of comprehensible input—language that is just beyond the learner's current proficiency level. Krashen's hypothesis posits that learners acquire language most effectively when they are exposed to content that stretches their abilities without overwhelming them. Additionally, listening strategies such as bottom-up (focusing on decoding sounds and words) and top-down (using context and prior knowledge to interpret meaning) play essential roles in the learning process. A balanced integration of these strategies in well-designed listening activities can significantly enhance a learner’s language skills.

However, conventional classroom instruction often lacks the capacity to provide such a nuanced and individualized approach to listening practice. This gap in personalized instruction has been increasingly addressed by the emergence of adaptive learning tools and artificial intelligence (AI)-based educational systems. These technologies, by analyzing student data, dynamically adjust content to suit the individual needs of each learner. For instance, adaptive systems can modify the difficulty of tasks in real-time, enabling students to engage in self-paced learning while ensuring they are continuously challenged at an appropriate level. Such systems have been shown to outperform static, one-size-fits-all educational methods, leading to higher levels of student engagement and improved learning outcomes.

Despite these advancements, there remains a significant disparity in the level of investment between STEM education technologies and those supporting language learning, particularly in the area of listening comprehension. While substantial funding continues to flow into the development of interactive STEM tools such as coding platforms, virtual laboratories, and AI-driven simulations, language education, especially in the realm of listening skills, remains underfunded and underserved. For example, a 2023 report by HolonIQ revealed that over 60% of global EdTech funding was allocated to STEM innovations, while less than 20% was directed towards language learning technologies. This financial imbalance has resulted in an evolution of highly advanced STEM educational tools, while language learning—especially the nuanced skill of listening comprehension—lags behind.

Although platforms like Duolingo and Rosetta Stone have made significant strides in gamified language learning, they typically prioritize vocabulary and grammar acquisition over listening skills. Furthermore, they often fail to offer personalized, real-time feedback based on the learner's specific progress and comprehension abilities. As a result, students may not receive the focused attention they need to develop strong listening skills, an area critical for success in real-world communication and academic achievement.

To bridge this gap, this research proposes the development of a listening-focused educational platform specifically designed for students in Grades 10 to 12. The system incorporates adaptive listening activities, validated by a machine learning model that uses cosine similarity to compare student responses with the correct reference answers. This model allows for greater flexibility in evaluating student comprehension by focusing on the semantic accuracy of their responses rather than requiring exact matches in phrasing. The system also incorporates predictive analytics and engagement monitoring, including facial expression recognition, to deliver personalized feedback tailored to each learner's emotional and cognitive states. By leveraging these advanced technologies, the platform aims to provide a dynamic, engaging, and student-centered learning experience that enhances comprehension and supports long-term language development.

Moreover, this research acknowledges the growing need to address issues of data privacy and ethical use of student information, particularly when working with minors. With the increasing use of AI and behavioral tracking technologies, concerns regarding the security of student data and the potential for misuse of personal information have become central to the conversation surrounding educational technology. This research prioritizes transparency and security, ensuring that student data is handled responsibly and in compliance with relevant data protection regulations, such as COPPA and GDPR.

Ultimately, the goal of this research is to contribute to the advancement of language learning technologies by offering a solution that not only enhances listening comprehension but also fosters a more personalized and adaptive learning environment for students. Through the integration of machine learning and AI-driven technologies, this platform aims to provide a more effective, engaging, and secure way for students to develop essential language skills, preparing them for success in both academic and real-world communication contexts.

**Listening Activities in Language Learning**

Listening activities are a fundamental component of second language learning, and numerous studies have explored how technology can enhance these practices. The evolution of digital tools has transformed traditional approaches to listening comprehension, creating more interactive and personalized learning experiences.

**Listening and Skill-Level Adaptation**

One critical area is listening and skill-level adaptation, where digital platforms automatically adjust task difficulty based on learner performance. Such systems often use multiple-choice questions or form-completion exercises, aligning with mastery learning principles that rely on formative assessment data to guide progress [2]. This adaptive approach represents a significant advancement over fixed-difficulty materials, as it creates a personalized learning path that responds to individual strengths and challenges.

The cognitive processes underlying listening comprehension involve both bottom-up and top-down processing working in tandem. Bottom-up processing requires listeners to decode individual sounds, words, and sentences to construct meaning—a particularly challenging task for language learners encountering unfamiliar vocabulary or speech patterns. Complementarily, top-down processing leverages contextual cuesand background knowledge toanticipate and interpret meaning, helping learners navigate through unclear speech segments. Effective listening platforms must support the development of both processing types to build comprehensive listening skills.

Schema Theory further illuminates how prior knowledge activation significantly impacts comprehension. When listeners can access relevant mental frameworks based on past experiences and cultural knowledge, they more efficiently process new auditory information. Educational technologies that activate appropriate schemas before listening tasks can dramatically improve comprehension outcomes, especially for content with unfamiliar contexts or specialized vocabulary.

**Structured Listening Activities**

Another approach is through structured listening activities, which involve academic lectures, scripted dialogues, and real-life scenarios. These activities are categorized by difficulty level and target specific listening sub-skills such as note-taking, comprehension, and inference-making. Structured activities ensure that learners interact with content appropriate for their current proficiency, promoting better retention and understanding [4].

The effectiveness of these activities depends significantly on working memory capacity, which plays a crucial role in listening comprehension. Working memory allows learners to simultaneously hold earlier parts of audio content while processing incoming information demanding cognitive task in a second language. When listening materials exceed working memory capacity, comprehension deteriorates rapidly. Consequently, well-designed platforms often feature shorter audio segments (1-1.5 minutes) that remain within cognitive processing limits, particularly for intermediate learners.

These structured activities typically incorporate various discourse types—including narratives, conversations, voicemails, location descriptions, and instructional content—each developing distinct listening skills. For example, narrative listening builds sequential comprehension and inference-making, while conversational listening develops turn-taking awareness and pragmatic understanding. This variety ensures comprehensive skill development across different communicative contexts.

**Authentic vs. Modified Materials**

The debate between authentic and pedagogically-modified materials continues to influence listening instruction design. While authentic materials expose learners to natural language use with realistic speech rates and pragmatic features, pedagogically-modified content offers scaffolded learning experiences that build confidence before tackling more challenging authentic speech. Many successful platforms incorporate both approaches, gradually transitioning from modified to authentic materials as learners progress.

Exposure to varied accents represents another crucial dimension in listening instruction. Most platforms focus primarily on standard American and British accents, potentially leaving learners unprepared for the diversity of English variations they will encounter in real-world settings. Research indicates that systematic exposure to accent variation significantly improves listening flexibility and overall comprehension in diverse communicative contexts.

**Technology-Enhanced Approaches**

The technological evolution in listening instruction has marked a significant shift from passive to active listening technologies. Traditional approaches often limited learners to passive reception of audio content with little control or interaction. Modern platforms encourage active engagement by allowing learners to pause content to process information at their own pace, while strategically limiting replay options to simulate real-world listening conditions where repetition isn't always available.

Additionally, there are interactive listening exercises that aim to boost engagement through gamification. Platforms like LyricsTraining offer transcription tasks synced with popular songs, reinforcing listening skills through rhythm and repetition. Other tools like BBC Learning English and TED-Ed provide access to authentic spoken English, often with transcripts and comprehension quizzes. Elllo.org, for example, features recorded conversations from speakers of different English accents, which helps students build listening flexibility.

**AI-Powered Assessment and Feedback**

Artificial intelligence is transforming listening instruction by enabling more sophisticated assessment and feedback mechanisms. Traditional assessment methods typically relied on multiple-choice questions with delayed feedback, limiting learners' ability to make immediate corrections. AI-powered systems now offer real-time, semantic assessment of responses through technologies like cosine similarity analysis, which measures conceptual accuracy rather than expecting verbatim answers. This approach allows for personalized feedback that addresses specific comprehension gaps while accommodating diverse response phrasings.

Empirical studies demonstrate the effectiveness of these adaptive listening platforms. Research by Wang et al. (2018) showed that students using adaptive listening technologies improved their listening accuracy by 25% over six weeks compared to control groups. Similarly, Yang (2016) reported a 35% improvement in comprehension scores among technology-enhanced learning groups, with particularly strong results in long-term retention. These statistical improvements highlight the potential of AI-driven platforms to accelerate listening skill development.

**Metacognitive Development**

Technology can significantly enhance metacognitive awareness in listening—the ability to monitor and regulate one's comprehension strategies. Platforms that provide immediate feedback encourage learners to reflect on their listening processes, recognize comprehension breakdowns, and adjust their approaches accordingly. This metacognitive development is crucial for autonomous learning, as it equips students with strategies to navigate challenging listening situations independently.

Self-assessment tools further support this development by enabling flexible, consistent practice outside formal instructional settings. The reliability and variety of these tools help learners identify specific strengths and weaknesses, leading to more targeted improvement strategies. Research indicates that regular self-assessment correlates strongly with improved listening proficiency over time, particularly when combined with adaptive content delivery.

**Cultural and Contextual Dimensions**

Cultural context significantly influences listening comprehension, as language is inseparable from cultural norms, references, and communication patterns. Without adequate cultural knowledge, learners may correctly hear words but misinterpret meanings, especially with idiomatic expressions, humor, or culturally-specific references. Technology can bridge these cultural understanding gaps by providing contextual information through pre-listening activities, cultural notes, and explanatory glossaries that illuminate cultural nuances.

More advanced platforms deliver contextual knowledge through multimedia resources that illustrate cultural settings and communication dynamics. By integrating authentic materials from diverse cultural contexts, these platforms help learners develop cultural awareness alongside linguistic comprehension, preparing them for real-world communication across cultural boundaries.

**Gaps in Current Approaches**

While these tools are valuable, many still lack real-time adaptability and personalized feedback based on a learner's strengths or weaknesses. Current platforms often fail to integrate cognitive processing models, metacognitive strategy development, and cultural context in cohesive ways. There remains a significant gap between research findings on effective listening instruction and their implementation in widely available learning technologies.

Additionally, most platforms do not adequately track long-term skill development or facilitate transfer of listening strategies across different contexts. Without these features, learners may develop skills in isolated exercises without building the comprehensive listening proficiency needed for authentic communication. Future research must address these limitations to create more holistic, evidence-based listening instruction platforms.

### Research Gap

---add the table here—

**Current Market Limitations**

Popular language learning platforms like Duolingo and ELLLO have made significant contributions to language education but exhibit notable limitations in specialized listening comprehension development. These platforms primarily focus on general language acquisition, offering a broad range of activities targeting vocabulary building, grammar, and basic conversational skills. While they include listening components, they do so in service of overall language exposure rather than developing specialized listening comprehension skills.

For example, Duolingo employs short audio clips tied to sentence translation or matching tasks, but these clips are typically simplified and lack the complexity of real-life listening situations. Similarly, ELLLO provides authentic audio from global speakers, yet its primary emphasis is on everyday conversations and cultural exposure rather than structured listening strategies like inference-making, note-taking, or identifying main ideas in academic or professional contexts. As a result, these platforms serve well for general language practice but fail to adequately prepare learners for advanced listening tasks requiring deeper cognitive processing.

Furthermore, widely used language learning platforms lack deep personalization when it comes to individual listening progress and needs. While they may offer some adaptive learning features, such as adjusting difficulty after a correct or incorrect response, they typically do not provide detailed, personalized feedback based on specific listening sub-skills like comprehension, inference, or vocabulary recognition. These platforms generally follow a linear or pre-set curriculum that doesn't fully account for a learner's unique challenges or strengths in listening comprehension. Consequently, learners may repeatedly encounter content that is either too easy or too difficult, without targeted guidance to improve weak areas, limiting the efficiency and effectiveness of their listening practice.

Additionally, platforms like Duolingo and ELLLO demonstrate a limited ability to adapt content difficulty based on real-time performance. While some basic adaptive features exist—such as advancing to harder levels after correct answers—they often lack the capacity to adjust listening material dynamically as the learner progresses through an activity. These platforms do not analyze listening accuracy in real-time or modify the complexity of audio clips, vocabulary, or question types based on a student's ongoing performance. This shortcoming creates learning experiences that may be either overly challenging or insufficiently stimulating, reducing the overall effectiveness of listening practice sessions.

**Pedagogical Framework**

Our proposed system addresses these gaps by implementing established pedagogical theories that current platforms largely overlook. Unlike many existing platforms that follow a one-size-fits-all approach, our system incorporates established pedagogical theories such as Mastery Learning and Cognitive Load Theory to provide a more personalized and effective listening experience. Through Mastery Learning, learners are only allowed to progress once they demonstrate a solid understanding of each listening task, ensuring deep comprehension rather than surface-level familiarity. Simultaneously, Cognitive Load Theory informs the design of our activities by using short, focused audio clips (around 1 to 1.5 minutes) to avoid overwhelming the learner's working memory. This approach supports sustained attention and better retention, whereas platforms like Duolingo or ELLLO often lack these theoretical underpinnings and do not adapt content based on learner mastery or cognitive capacity, limiting their ability to support long-term skill development.

Furthermore, our system incorporates well-established pedagogical theories such as Scaffolded Learning and Vygotsky's Zone of Proximal Development (ZPD) to enhance listening skill development. It features two types of sessions: Main Session and Practice Session. In the Main Session, the system automatically assigns listening activities based on the student's current proficiency level. This adaptive approach ensures learners are constantly working within their ZPD — tasks are challenging but achievable with some effort, promoting steady progress. Upon completing the Main Session, a student's level is incrementally increased, reflecting their growth. The Practice Session, on the other hand, allows students to self-select a difficulty level (easy, medium, or hard), supporting scaffolded learning by giving them autonomy to reinforce skills at their own pace without affecting their progression level. This structure balances guided development with independent practice, aligning with core educational models often overlooked by other systems.

**Technology Integration**

Our proposed system leverages cutting-edge technology to overcome the limitations of existing solutions. It uses Natural Language Processing (NLP) and Machine Learning (ML) algorithms not only to evaluate student answers in real time but also to assist in automatically generating questions based on the audio content. NLP techniques help extract key information such as keywords, phrases, and sentence structures from audio transcripts, which are then used to create context-relevant questions. ML models ensure that the questions are appropriate for the learner's level, supporting personalized and adaptive learning experiences that current platforms fail to provide.

The system's approach to processing and analyzing student responses represents a significant advancement over existing platforms. Student responses are compared with the original answers using a machine learning model that calculates a similarity score. This score helps evaluate the student's listening comprehension performance. If the student performs well, the system increases the difficulty level in the next main session. However, if the performance is low, the system maintains the difficulty level to allow more practice at the current level. This approach ensures that content adapts to the learner's progress without overwhelming them, creating a more personalized and effective learning experience than what is currently available.

**User Experience Details**

Our proposed system offers significant improvements across all features identified in the comparative analysis table:

1. **Listening Activities:** While Duolingo and ELLLO offer various listening activities including sentences, phrases, and conversations that primarily focus on general language acquisition, our system provides tailored listening activities based on the student's proficiency level. The system delivers short, contextualized listening tasks where students must answer questions based on audio clips, promoting deeper comprehension. These tasks dynamically adjust based on the learner's progress, creating a more personalized and effective learning experience.
2. **Listening Practice Activities:** Both Duolingo and ELLLO include practice exercises focusing on vocabulary and grammar, with listening tasks integrated into overall learning activities. In contrast, our system offers dedicated practice activities where students can select difficulty levels (easy, medium, or hard). These practice sessions are separate from main sessions, allowing students to focus on areas of difficulty without affecting their progression in the main activity. This separation enables targeted practice without the pressure of advancement, supporting more comprehensive skill development.
3. **Question Generation Based on Skill Level:** Neither Duolingo nor ELLLO adjusts question difficulty based on learner performance in real-time. Our system uses machine learning algorithms to assess student responses and generate questions matching their current skill level. For instance, if a student performs well in a set of listening activities, the system automatically increases the difficulty of subsequent tasks to provide appropriate challenge. If performance is low, the system maintains the same difficulty level to allow for reinforcement and mastery, ensuring optimal learning conditions at all times.

**Stakeholder Benefits**

The proposed system offers substantial benefits to various stakeholders in the language learning ecosystem:

**Students:**

Students receive a personalized learning experience as the system adapts content difficulty based on their real-time performance. This tailored approach allows progression at an individual pace, ensuring thorough understanding before advancement. The ability to select practice sessions of varying difficulty levels provides greater control over learning, enabling reinforcement or self-challenge as needed. Immediate scoring and detailed feedback create an effective feedback loop that facilitates better retention and faster concept mastery. Additionally, the level advancement system provides a sense of achievement and progression that maintains motivation and engagement throughout the learning process.

**Teachers:**

The system generates detailed performance reports that allow teachers to quickly identify struggling areas for individuals or entire classes. This data-driven approach enables targeted support and teaching strategy adjustments. Teachers can focus their attention on students requiring extra help while allowing others to progress independently, reducing administrative tasks and enabling concentration on higher-level teaching. The system's automation capabilities allow teachers to scale their efforts to accommodate larger student numbers without compromising feedback quality or individualized attention.

**Institutions:**

Educational institutions benefit from a cost-effective solution that provides personalized learning and feedback while reducing the need for constant one-on-one interventions. The system can simultaneously support numerous students without additional resources. By adopting these adaptive learning technologies, institutions can improve student outcomes, leading to higher achievement and graduation rates. The data generated by the system supports informed decisions about curriculum design, resource allocation, and instructional strategies, ensuring alignment with actual student learning needs.

**Measurable Learning Outcome Improvements**

Addressing the identified gaps directly contributes to measurable improvements in learning outcomes:

The personalization features ensure students are continuously challenged at an appropriate level, preventing both overwhelm and boredom. This optimization of difficulty leads to more efficient concept mastery and longer information retention, resulting in measurable improvements in listening comprehension skills. The system's instant feedback helps students identify and correct misunderstandings immediately, enhancing retention and deepening learning. Continuous progress tracking through scoring visualizes skill development, motivating continued improvement and resulting in higher test scores and better overall performance over time.

The system's skill-level adjustments allow students to progress at their own pace, ensuring mastery before advancement. This approach aligns with mastery learning principles, leading to greater retention and deeper understanding. As students advance only upon demonstrating proficiency, the quality of learning improves, resulting in fewer knowledge gaps and measurable improvements in listening comprehension and language proficiency.

Increased student engagement through visible progression and practice session difficulty choices encourages sustained effort. Motivated students invest more time and practice, resulting in measurable improvements in listening comprehension and overall language skills. For teachers, the system's data provides clearer insights into student progress, helping identify struggling areas and enabling targeted interventions. This tailored support leads to improvements in comprehension skills across the student population.

The system's scalability allows institutions to extend their learning efforts while maintaining personalized feedback. This ensures all students receive necessary support regardless of class size. As institutions track progress across larger student groups, they can observe improvements in group-level outcomes, such as higher average scores on listening comprehension assessments and increased retention rates.

### Research Problem

An essential feature of educational technology, especially in a customized education environment, is to offer content and support suitable for each student. In traditional learning methods, it is often challenging to personalize assignments based on individual student needs and skill levels, particularly when dealing with complex tasks such as listening comprehension. Current systems often provide generalized content that doesn't account for the varying levels of proficiency among learners. This lack of personalization can limit student progress and engagement, as each student may require different levels of support and difficulty to optimize their learning experience. To address these challenges, the listening activity system must be able to accurately assess the student's current skill level and dynamically adjust the difficulty of assignments and practice sessions accordingly. By doing so, learners will be able to engage with content that aligns with their individual needs, ensuring a more effective learning process. Therefore, this research will focus on several key questions that will guide the development of a more adaptive and personalized system:

1. How can the system align listening assignments to individual students and optimally design practice sessions?
2. How do you ensure that questions are appropriately aligned and personalized to cater to individual learners?
3. In what ways can providing informal opportunities for students to practice their listening skills contribute to more structured and formal learning activities?
4. How can the system ensure that students have continuous, relevant support when and where they need it to enhance their listening comprehension and overall English learning experience?

### Objectives

#### Main Objectives

The primary aim in this investigation is to create an integrated LMS listening activity system for 10–12 years old students that could be conveniently used with the personalized learning approach. Our vision is to build a system which can evaluate the levels of proficiency and suggest custom listening activities as well as practice sessions, ultimately enhancing students' learning experiences.

This research seeks to develop a comprehensive, data-driven solution that bridges the gap between standardized listening exercises and individualized learning needs. By implementing advanced adaptive algorithms and user-centered design principles, the system will continuously evolve to meet each student's unique learning trajectory, fostering greater engagement, confidence, and measurable improvement in listening comprehension skills. The platform aims to transform passive listening activities into interactive, personalized learning journeys that respond dynamically to student performance while maintaining alignment with educational standards for this critical age group.

#### Specific Objectives

**Question Collection for Main Listening Activities and Practice Sessions (Data Sets):**

1. Collaborate with the supervisor to curate appropriate question sets for students aged 10 to 12. Get questions according to finding location, conversation, voice mails, story telling and giving instructions.
2. Ensure that the question sets are available in varying levels of difficulty: easy, medium, and hard.
3. Develop a comprehensive tagging system for all questions that captures linguistic complexity, vocabulary level, cultural context, and thematic content to enable precise matching with student profiles.
4. Create a validation protocol to ensure all content is developmentally appropriate and aligned with educational standards for 10-12 year olds.
5. Establish a continuous content refresh cycle to maintain engagement through new material additions on a regular schedule.

**Student Skill Level Assessment:**

1. Integrate and retrieve student skill levels from the LMS system, which are calculated based on their performance in other components.
2. Use these skill levels to tailor the main listening activities to meet individual student needs.
3. Develop a multi-dimensional proficiency model that evaluates separate listening sub-skills including comprehension, inference, vocabulary recognition, and cultural understanding.
4. Implement a baseline assessment mechanism for new students with limited history in the system.
5. Design a real-time skill level adjustment algorithm that responds to performance patterns across multiple sessions rather than single-instance results.

**Practice Session Development:**

* 1. Design and implement practice sessions, including multiple rounds, to help students prepare for the listening activities.
  2. Provide tutorials and guidance to support students in navigating and successfully completing the listening activities.
  3. Create a scaffolded practice pathway that gradually increases complexity while maintaining student confidence.

**User Interface (UI) Design:**

* 1. Design a clean, visually aesthetic, and user-friendly interface that is easy for students to navigate.
  2. The UI must be appropriate for children, engaging, and intuitive so that kids can navigate the site quickly.
  3. Conduct usability testing with the target age group to validate design choices and navigation patterns.
  4. Create a simple interface that makes learning accessible to all students.
  5. Use bold colors, juvenile fonts, and interactive elements that attract children's attention.
  6. Make it user-friendly and easy to navigate, focused on the young learners.
  7. Implement an intuitive answer submission system that minimizes frustration when entering text responses.

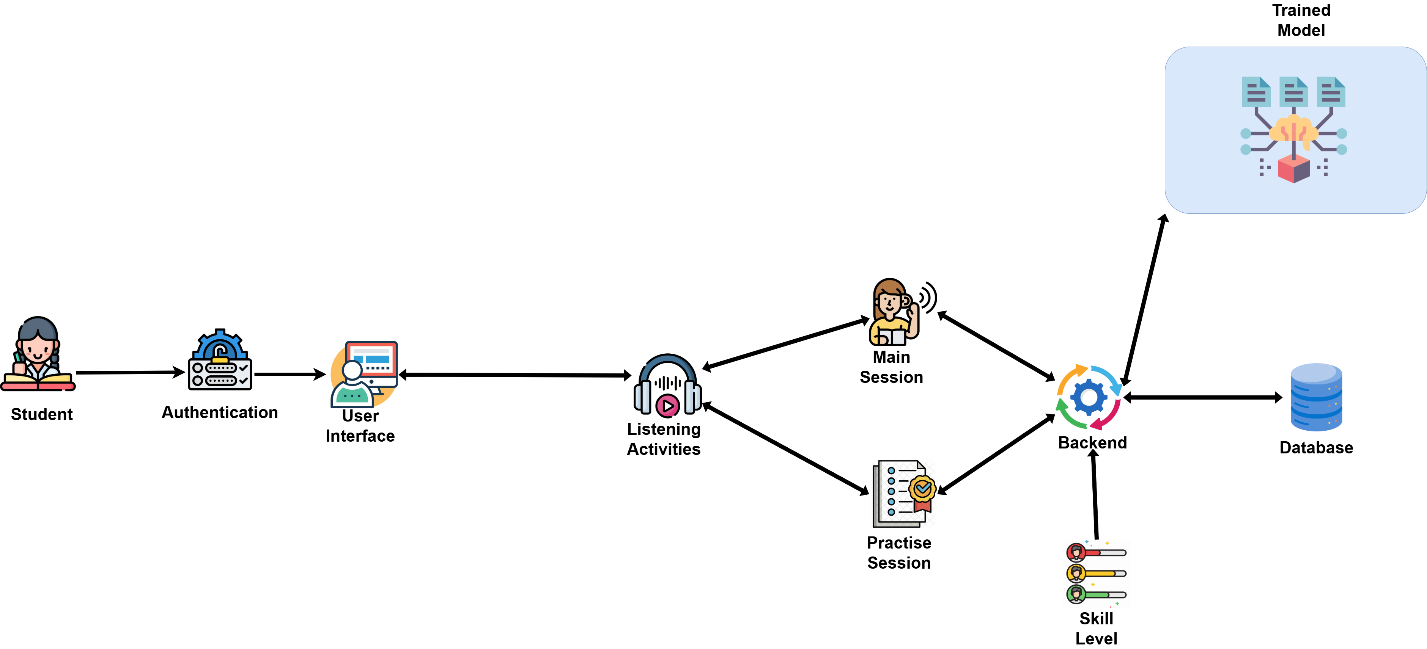
**Get Suitable Educational Content through the Supervisor:**

* 1. Interview the supervisor and collect appropriate educational data with 10–12-year-old student-specific content.
  2. Make sure the content maps to learning objectives and curriculum.
  3. Establish a quality assurance framework for content selection that evaluates cultural relevance, linguistic appropriateness, and engagement potential.
  4. Develop protocols for regular content reviews and updates based on student performance data and educator feedback.

# Methodology

#### Methodology

This research adopts a technology-enhanced approach to improve listening comprehension skills among Grade 10–12 students. The proposed system is structured around two main learning modes: the Main Session and the Practice Session. Both modes are built on a web-based client-server architecture, consisting of a frontend interface for student interaction, a backend server for processing and data handling, and a database for storing learning materials, user data, and assessment results. Additionally, a cosine similarity-based machine learning model is used to evaluate student responses in the Main Session, providing adaptive feedback and performance tracking.



The proposed system is designed as a personalized Learning Management System (LMS) that adapts listening activities based on the proficiency level of each student. The system is structured into two distinct modes: the Main Session and the Practice Session. In the Main Session, the system dynamically recommends listening activities based on the student’s current skill level. These recommendations are retrieved from a MongoDB database, which contains metadata for each listening activity. Each record in the database includes essential details such as the title, difficulty level (which is linked via a MongoDB ObjectId), category ID, audio file URL, and associated question IDs.

When students begin their session, they first choose a category such as “Finding Location,” “Conversation,” “Voicemails,” “Storytelling,” or “Giving Instructions.” The system then filters the available listening activities based on their selection and displays a list of relevant exercises. Once the student picks a listening activity, the React frontend sends a REST API request to the Node.js backend, which returns detailed data for that particular activity, including the audio file location, questions, and related metadata. This triggers a navigation to the listening page, where the student can begin the activity.

Once on the listening page, the student is presented with a start button that initiates a countdown. When the countdown reaches zero, the system streams the selected audio directly from cloud storage, using the path provided in the database. After the audio playback concludes, the system automatically displays five comprehension questions, which are linked to the selected listening activity via MongoDB document references. The student can then interact with each question, typing their answer in a pop-up container, and submitting all answers by pressing the “OK” button.

Following the submission, the system transitions to a results page where the student’s answers are evaluated by a cosine similarity model. This model, written in Python and deployed as a Flask API, accepts the student’s answers and the expected answers as JSON objects. The responses are vectorized into 26-dimensional vectors based on character frequencies of letters a-z. Cosine similarity is then calculated between these vectors, providing a score that represents how closely the student’s answer matches the expected answer. This similarity-based evaluation is designed to be both efficient and tolerant of minor variations in spelling or word choice, making it an ideal choice for this task. For example, responses like "at the meeting" and "in the meeting" will score highly due to their similarity in character composition, even though the phrasing is slightly different.

The model returns a similarity score for each question, ranging from 0 to 1, and the overall score is calculated based on these individual scores, yielding a percentage result. If the student’s current score exceeds their previous highest score for the same activity, the result is recorded and sent to a separate progression module. This module is responsible for tracking student progress and updating their proficiency level based on predefined criteria. The level progression then influences future activity recommendations, ensuring that students are consistently presented with appropriately challenging content. All performance data, including question answers and similarity scores, is stored in MongoDB to be used for future recommendations and adaptive learning.

In the Practice Session, students can manually select a difficulty level and freely explore listening exercises without the added pressure of scoring or level progression. The Practice Session mimics the user experience of the Main Session, including category selection, audio playback, and question answering. However, unlike the Main Session, there is no similarity-based scoring or level progression in the Practice Session, making it an ideal environment for risk-free exploration and skill reinforcement.

The system architecture consists of a React frontend, a Node.js backend, and a MongoDB database. Communication between components is managed through REST APIs, ensuring seamless integration between the frontend, backend, and the machine learning evaluation model. Audio files are stored in cloud storage rather than directly in the database, keeping the system lightweight and scalable. The frontend is fully mobile-responsive, featuring four main pages: a homepage for mode selection, a category page, a listening interaction page, and a results page. The user interface is designed with simplicity and usability in mind, featuring progress bars during audio playback, clean layouts, and easy navigation.

The MongoDB database stores all the system’s data, including collections for listening activities, questions and answers, categories, and difficulty levels. Each listening document includes a reference to the audio file URL, category ID, difficulty level, and an array of references to associated questions and answers. The question and answer documents contain the original text of the questions and the expected answers, while the category collection holds metadata such as the name, description, and background images for each category. The difficulty level collection stores information such as the difficulty’s name, numeric weight, and an availability flag to indicate whether the difficulty level is currently active or not.

In terms of security and privacy, the platform ensures that student data is stored and transmitted securely. The authentication system, developed by a separate team, ensures secure user login and accounts for privacy concerns. No personally identifiable information (PII) is stored alongside student responses, which are stored in plain-text format within MongoDB. Secure transmission protocols are used to encrypt student data as it is exchanged between the frontend, backend, and machine learning model, ensuring compliance with educational data privacy standards.

Feedback is provided to students at the end of each session, allowing them to see whether their answers were correct, alongside the original answer for transparency. The feedback is presented as a holistic review of performance rather than a question-by-question breakdown. This approach helps students understand their overall progress and areas for improvement. The system retains the highest score for each listening activity, ensuring that students’ progress is tracked accurately without penalizing them for earlier attempts. This score is passed to the progression system, which updates the student’s level accordingly, ensuring that future Main Sessions provide increasingly challenging content tailored to the student’s proficiency.

The system undergoes unit testing and user testing as part of its development process. However, full-scale testing and deployment are planned for later phases, once the core functionality has been thoroughly verified. Given the modular architecture of the system, it is easy to extend or adapt to new features in the future. This could include the integration of speech recognition to allow students to answer questions by speaking or the addition of advanced analytics to track student performance in more detail.

This methodology emphasizes a lightweight, adaptive, and scalable design for the LMS, ensuring that it is capable of providing a personalized learning experience that is both engaging and effective for students. By integrating machine learning for automated evaluation and providing dynamic content recommendations based on student progress, the system aims to enhance listening skills in a way that adapts to each learner’s needs.

#### Testing & Implementation

Implementation

Word Similarity model

To evaluate student responses against the expected answers in a lightweight and efficient manner, a custom cosine similarity model was developed using Python. This model is designed to measure lexical-level similarity based on character frequency, offering a fast, language-independent method suitable for educational environments where real-time feedback is essential.

The similarity checking component is separated into two Python files:

* **word\_similarity.py** – Contains the core logic for calculating cosine similarity between two strings.
* **app.py** – A Flask-based REST API that exposes the similarity model to the main system for integration.

**Word\_similarity.py**

#### **Libraries and Tools Used**

To implement the model, the following Python libraries were used.



Figure 1. Model Libraries

**1. NumPy**

The NumPy library was employed to create and manipulate numerical arrays specifically, vectors that represent the frequency of characters in input strings. A zero-initialized vector of length 26 was created to represent the English alphabet (a–z). This approach allows for efficient character-level representation of words, where each character corresponds to a unique index in the vector.

* np.zeros(len(chars))

this line initializes a vector filled with zeros, where chars is a string containing the 26 lowercase English letters. This zero vector is used as a base to populate character frequencies from the input text. The choice of NumPy over native Python lists stems from its speed and memory efficiency. NumPy arrays are implemented in C and support vectorized operations, allowing computations to be done over entire arrays without explicit loops. This is especially beneficial in real-time systems like the proposed LMS, where many comparisons may be required in quick succession as students submit their answers.

Each character frequency vector was then reshaped into a 2D array using

* vector.reshape(1, -1)

This step converts a 1D array into a 2D array with one row and 26 columns. The reshape is necessary because the cosine\_similarity() function from scikit-learn expects inputs to be in a 2D matrix format each row representing a separate observation. For example, a single word like "cat" becomes a row vector like [1, 0, 1, ..., 1, ..., 0], where only relevant character indices (for 'c', 'a', and 't') are incremented. Without this reshaping, the similarity function would throw an error or misinterpret the vector structure.

Internally, NumPy acts as a bridge between the text input and the numerical analysis. It converts character counts into a structured, fixed-size format and prepares it for similarity evaluation. This integration ensures that the character frequency data is both lightweight and computationally optimized for further processing.

**2. collections.Counter**

To generate character frequency vectors from text, the built-in Counter class from Python’s collections module was used. This class converts a given string into a frequency dictionary, mapping each character to the number of times it appears. For example.

* Counter("three") = {'t': 1, 'h': 1, 'r': 1, 'e': 2}

Using this structure, it becomes simple to map the characters in a word or phrase to the corresponding index in the frequency vector. This enables fast and structured vectorization of input strings without the need for manually iterating through each character.

Before the Counter is applied, all input strings are converted to lowercase to ensure consistency and to limit the frequency vector space to 26 positions—each representing a lowercase letter from 'a' to 'z'. Non-alphabetic characters such as punctuation marks, digits, or symbols are filtered out during the vector construction process to maintain a focused and lightweight feature set.

This method is applied not only to single words but also to full phrases. For instance, a student answer like "in the meeting" is treated as one string, and Counter calculates the character frequency across the entire phrase. This helps retain the structural essence of the sentence while abstracting away the need for token-based (word-level) comparisons.

The Counter object is then used to populate a zero-initialized NumPy vector of length 26. Each character’s frequency is inserted into its appropriate index using the ASCII offset: ord(char) - ord('a'). This mapping ensures, for example, that 'a' goes to index 0, 'b' to 1, and so on, up to 'z' at index 25. The final result is a fixed-length vector that numerically represents the composition of the input string in terms of character distribution.

Using Counter in this context contributes to the overall efficiency and simplicity of the model. It abstracts low-level frequency counting, integrates cleanly with NumPy, and aligns well with the requirements of the cosine similarity function used for answer evaluation.

**3. Scikit-learn’s cosine\_similarity**

After the vectors are constructed, the **cosine similarity** between the original answer and the student's answer is calculated using the cosine\_similarity() function from the scikit-learn library:

This function computes the angle-based similarity between two vectors using the formula

* =
* and are the character frequency vectors
* ⋅ is the dot product
* ∥ ∥ and ∥∥are the magnitudes of the vectors.

This returns a similarity value between **0 and 1**

* **1.0** indicates that both answers have identical character composition.
* **0.0** indicates complete dissimilarity in character composition.

This vector-based approach enables robust matching that tolerates minor spelling errors or word reordering, making it ideal for listening-based language assessments. For instance, it can rate "at the meeting" and "in the meeting" as highly similar due to their nearly identical character composition.

#### **Functions and Methods**

1. **preprocess\_word**

* This function is a simple but essential preprocessing step in the similarity evaluation pipeline. It takes a single word or phrase as input and returns its lowercase version

A black rectangular object with white text

AI-generated content may be incorrect.

Figure 2. Model Method 1 preprocess\_word

* In natural language processing (NLP) especially in tasks like text comparison or vectorization, consistency in text formatting is crucial. This function ensures that all input strings are converted to lowercase before further processing such as character frequency analysis or vectorization. For instance.
  + "Three" becomes "three"
  + "IN THE MEETING" becomes "in the meeting"
* Without this step, characters like 'T' and 't' would be treated as different elements, which would distort the frequency vector and reduce the accuracy of cosine similarity calculations. Since the system only tracks characters a through z, upper-case letters would otherwise be ignored or mapped incorrectly.

1. **vectorize\_word**

* This function plays a central role in transforming a word or phrase into a fixed-size numerical vector, making it suitable for cosine similarity comparison.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 3. Model method vectorize\_word

* This function converts any given string (typically preprocessed to lowercase beforehand) into a 26-dimensional vector, where each dimension represents the frequency of a specific English alphabet character (from 'a' to 'z').
* For example, the word "cat" will be converted into a vector like
  1. [1, 0, 1, 0, ..., 1, ..., 0]
* Here, positions corresponding to 'a', 'c', and 't' have the value 1, and all others are 0.
* **Breakdown**
  1. **Character counting with Counter**
     + char\_counts = Counter(word)
     + This creates a dictionary-like structure containing each character's frequency in the input string.
       - Example: "cat" to {'c': 1, 'a': 1, 't': 1}
  2. **Initialize a zero vector**

chars = 'abcdefghijklmnopqrstuvwxyz'

vector = np.zeros(len(chars))

* + - A NumPy array of size 26 is created, with all values initially set to 0. Each index corresponds to a letter in the alphabet.
  1. **Populate the vector**

for i, char in enumerate(chars):

vector[i] = char\_counts.get(char, 0)

* + - The loop iterates through the alphabet. For each character, it checks if it appears in the input string using char\_counts.get(char, 0) and assigns the count to the corresponding position in the vector.
  1. **Return the vector**
     + The result is a structured numerical representation of the word that can be passed to similarity functions such as cosine\_similarity().

**3.****cosine\_sim**

* The cosine\_sim() function represents the core logic used to measure the similarity between the original answer and the student’s response. This function takes two string inputs—typically the expected and submitted answers—and returns a similarity score between 0 and 1 using the cosine similarity technique.
* Internally, the function first applies a preprocessing step by converting both input strings to lowercase. This step, handled by the preprocess\_word() function, ensures case insensitivity during comparison, meaning that "Meeting" and "meeting" are treated identically. After preprocessing, each word is passed to the vectorize\_word() function, which transforms the text into a 26-dimensional character frequency vector. Each vector corresponds to the number of times each letter from 'a' to 'z' appears in the string. These vectors are then reshaped into two-dimensional arrays using .reshape(1, -1) to match the input format expected by scikit-learn’s cosine\_similarity() function.
* The cosine similarity function then computes the angular similarity between these two vectors. If the characters in both strings are distributed similarly, the cosine value approaches 1, indicating a strong match. If the vectors differ significantly, the similarity approaches 0. The function finally returns a float score, extracted from the similarity matrix as similarity[0][0].
* For instance, comparing the phrases “at the meeting” and “in the meeting” might yield a high similarity score, such as 0.91, due to their overlapping character compositions. However, comparing “three” and “3” results in a score of 0, since the numeric character is excluded from the alphabetic vectorization scheme.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 4. Model method cosine\_sim

**App.py**

This file functions as the entry point for the Flask-based API that connects the machine learning model to the Node.js backend. Several key Python libraries are imported to support web functionality, cross-origin communication, and modular architecture.

#### **Libraries and Tools Used**

Below is an explanation of each library and its purpose within the system

A screen shot of a computer

AI-generated content may be incorrect.

Figure 5. App.py libraries

1. **from flask import Flask, request, jsonify**
   * The Flask module is the core framework used to create the lightweight web server for the machine learning model. It allows the application to define routes (/cosine-similarity) that listen for HTTP requests.
   * **Flask** Used to initialize the web app, **request** Used to retrieve the incoming JSON data from the client, **jsonify** Used to return structured JSON responses to the client, ensuring compatibility with frontend systems.
2. **from flask\_cors import CORS**
   * CORS (Cross-Origin Resource Sharing) is a security feature implemented in browsers that blocks frontend JavaScript code from accessing APIs hosted on a different domain or port.  
     By using flask\_cors.CORS(app), the API explicitly allows cross-origin requests specifically, allowing the React frontend to communicate with the Flask server without being blocked.
3. **import os and import sys**
   * These built-in Python libraries are used for interacting with the file system and managing import paths. os.path.join() and os.path.dirname(\_\_file\_\_): These are used to build a file path that points to the custom word similarity model directory (similarity\_model3), making the project more portable and less dependent on fixed file structures. sys.path.append(...): This temporarily adds the model folder to the system path, allowing you to import custom functions from your own Python files

#### **Functions and Methods**

1. **Import word similarity model**

A black rectangular object with white text

AI-generated content may be incorrect.

Figure 6. import word similarity

* + This code allows the system to locate and import the cosine\_sim function from the word\_similarity module, which is used to calculate similarity between student answers and expected answers.

1. **cosine\_similarity\_api**

A computer screen shot of a computer code

AI-generated content may be incorrect.

Figure 7. Model Cosine Similarity Api

* The code snippet defines a Flask API endpoint to calculate cosine similarity between student answers and expected answers. First, the Flask application instance is created with app = Flask(\_\_name\_\_), and CORS is enabled with CORS(app) to allow cross-origin requests from the frontend. The @app.route('/cosine-similarity', methods=['POST']) decorator specifies the endpoint for the API, which accepts POST requests. The function cosine\_similarity\_api() handles incoming requests to this endpoint.
* Inside the function, the request.get\_json() method retrieves the JSON data from the request body, and the data is printed for debugging purposes with print(f"Received data: {data}"). The student and expected answers are extracted from the JSON body with data.get('word1') and data.get('word2'). The next step checks that both word1\_list and word2\_list are non-empty and that their lengths match, ensuring that each student answer has a corresponding expected answer. If these conditions aren't met, the function returns a 400 Bad Request response with an error message.
* If the validation is successful, the code loops through both lists using zip(word1\_list, word2\_list) and computes the cosine similarity between each pair of answers using the cosine\_sim(w1, w2) function. Each result, including the student answer, expected answer, and similarity score (rounded to four decimal places), is stored in a results list. The results are printed for debugging and then returned as a JSON response.
* In case of any errors during the process, the try-except block catches exceptions and returns a 500 Internal Server Error response with the error message. This API is crucial for calculating similarity scores in real-time, allowing the system to evaluate student responses efficiently.

Backend

The backend of the system is developed using Node.js, providing a scalable and efficient runtime environment for handling asynchronous operations. It plays a central role in managing communication between the frontend interface, the Flask-based machine learning model, and the MongoDB database. Key responsibilities of the backend include handling user authentication, managing API requests and responses, and delivering categorized listening materials such as finding locations, conversations, voicemails, storytelling, and giving instructions. It also orchestrates the evaluation process by receiving original and student-submitted answers from the frontend, forwarding them to the ML model, and retrieving the computed similarity scores. If a student achieves a higher score than their previous attempt, the backend updates their progression status accordingly and relays this to the appropriate module to adapt the learning level. Supporting tools like dotenv and CORS are used to manage environment configurations and enable secure cross-origin resource sharing. Through this structured flow, the backend ensures seamless data exchange and intelligent feedback delivery, forming a critical component of the adaptive learning experience.

**Folder Structure**

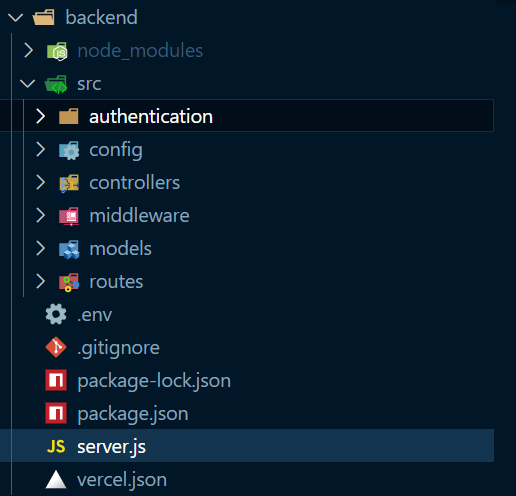


Figure 8. Backend Folder Structure

The backend of the system is structured in a modular way to ensure maintainability, scalability, and clear separation of concerns. At the root level, the main server file server.js initializes the Express server, loads environment variables, connects to the MongoDB database, sets up middleware, and registers all route handlers. The .env file holds sensitive configuration values such as database URIs and secret keys, keeping them isolated from the core codebase. Within the src directory, several subfolders organize core functionalities. The authentication folder contains authController.js, which handles user registration and login, including password hashing and token generation. The config folder includes env.js for loading environment variables and db.js for establishing the database connection using Mongoose. The controllers folder contains five key controllers: categoryController.js for managing category data, difficultyLevelController.js for difficulty levels, listeningController.js for managing listening activities and their metadata, qnaController.js for handling questions and answers including similarity evaluation requests, and userController.js for CRUD operations on user data.

The middleware folder contains authMiddleware.js, which authenticates API requests by verifying JWT tokens, ensuring secure access to protected routes. The models folder includes Mongoose schemas for each major entity: categories, difficulty levels, listening activities, QnAs, and users. These models define the structure of documents stored in MongoDB and enable interaction with the database. The routes folder defines REST API endpoints for each domain in the system. authRoutes.js handles login and registration, while the other route files (categoryRoutes.js, difficultyLevelRoutes.js, listeningRoutes.js, qnaRoutes.js, and userRoutes.js) expose endpoints for managing respective data, all protected by authentication middleware where necessary. This organized backend architecture enables clean API development, efficient data handling, and seamless integration with the frontend and the machine learning model for real-time answer evaluation.

**Libraries**

A black screen with white text

AI-generated content may be incorrect.

Figure 9. Backend Libraries

In the backend of the application, several essential libraries are utilized to ensure a functional, secure, and scalable server environment. The express library is used to create and manage the web server. Express is a lightweight and flexible Node.js framework that simplifies the creation of APIs and route handling, making it ideal for building scalable web applications. The cors library is included to enable secure communication between the backend and frontend, especially when they are hosted on different domains or ports. This is particularly important in modern web development where frontend and backend are often separated. Additionally, the dotenv library is used to load environment variables from a .env file into process.env. This allows sensitive configuration data such as database connection URIs, secret keys, and port numbers to be stored securely outside of the codebase, promoting better security practices and easier configuration management across different environments.

**Starts the backend**

A screen shot of a computer

AI-generated content may be incorrect.

Figure 10. Start the backend

This block of code is responsible for starting the server and allowing it to listen for incoming requests on a specific port. The line const PORT = process.env.PORT || 3000; checks if a port number has been defined in the environment variables (usually stored in a .env file). If a custom port is defined (e.g., PORT=3001), the server will use that; otherwise, it will default to port 3000. The app.listen(PORT, () => { }) function then starts the server on the specified port. Once the server is running, it executes the callback function and logs a confirmation message to the console, such as "Server is running on <http://localhost:3000>". This message helps developers know that the server has successfully started and is accessible via the given URL. In summary, this piece of code initializes the backend application and makes it ready to accept HTTP requests on the defined port.

**Middleware**

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 11. Backend Authmiddleware

The authentication middleware in this system is a crucial security component designed to protect sensitive routes by ensuring that only authenticated users can access them. This middleware checks each incoming request for a valid JSON Web Token (JWT), which serves as proof of a user’s identity. When a request reaches a protected endpoint, the middleware looks for the Authorization header containing the JWT, typically formatted as "Bearer <token>". If no token is found, the middleware immediately rejects the request with a 401 Unauthorized response, preventing unauthenticated access. If a token is present, the middleware removes the "Bearer" prefix and attempts to verify the token using a secret key (JWT\_SECRET) stored in the .env file. It then checks whether the token has expired by comparing the token’s expiration time (exp) against the current system time. If the token is invalid or expired, access is again denied. However, if the token is valid and current, the decoded user information is attached to the request object (req.user) for use in subsequent processes. The middleware then calls next() to pass control to the appropriate route handler. This layered security mechanism ensures that only users with valid credentials are able to interact with protected routes, maintaining the integrity and confidentiality of user data throughout the system. And this authentication middleware will used in every controller routes

Example

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12.Auth usage for routes

**Retrieve Categories**

A computer screen shot of a black screen

AI-generated content may be incorrect.

Figure 13. Get All Categories Method

This method is responsible for retrieving all the category records from the database and sending them to the frontend interface, allowing the system to dynamically display the list of available listening categories. The function getAllCategories is an asynchronous controller method, which uses Category.find()—a Mongoose method—to query the database and return all documents within the Category collection. If the data retrieval is successful, it responds with a status code 200 and returns the list of categories in JSON format. This response is then utilized by the frontend to populate category-related interfaces such as dropdowns or filter lists in the user interface. If an error occurs during the database operation (such as connectivity issues or query failures), the function catches the error and returns a 500 status code along with the error message, ensuring the system handles failures gracefully.

**Retrieve Listenings according to category**

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 14. Get Listening By Category

The getListeningsByCategory method is used to fetch all listening activities that belong to a specific category, based on the categoryId selected by the student on the frontend. This controller function is asynchronous and uses the Listening.find() method from Mongoose to query the database for all Listening documents where the category field matches the provided categoryId from the request parameters. To provide comprehensive data to the frontend, the function also uses the .populate() method to retrieve and embed the related QnA (questions and answers), category, and difficultyLevel objects for each listening item. This ensures that the response includes not only the listening audio details but also the associated metadata required for activity rendering and evaluation. If no listenings are found for the selected category, the method returns a 404 status code with a message indicating the absence of relevant data. On successful retrieval, a 200 status is returned along with the list of matched listenings in JSON format. If an error occurs during the database operation, a 500 status code is returned with an appropriate error message.

**Retrieve Listenings according to Difficulty**

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 15. Get Listening By Difficulty

The getListeningsByDifficulty method is responsible for retrieving all listening activities that match a specific difficulty level, identified by the difficultyLevelId received from the request parameters. This function plays a crucial role in both the Main Session and Practice Session of the system. In the Main Session, it is used to automatically present students with listening tasks that are aligned with their current proficiency level. In the Practice Session, it allows students to manually select a preferred difficulty level for self-paced learning. Internally, the method queries the Listening collection using Mongoose's find() function to retrieve all documents where the difficultyLevel field matches the provided ID. It also uses .populate() on related fields QnA, category, and difficultyLevel—to include detailed, connected information in the response. If no listening activities are found for the given difficulty level, the function returns a 404 status code with an appropriate message. Otherwise, it responds with a 200 status and the matched data in JSON format. In case of any errors during execution, a 500 error response is returned. This method enhances the personalized learning experience by ensuring that students engage with content suited to their individual learning needs and goals.

**Retrieve Listenings according to Difficulty and Category**

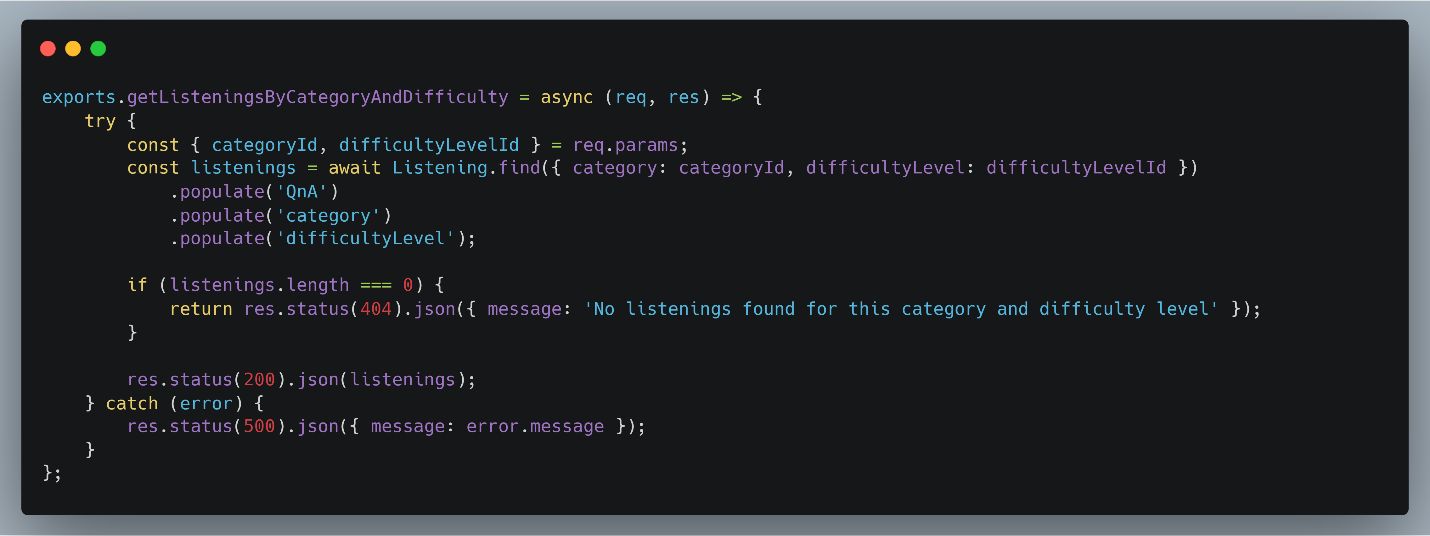


Figure 16. Get Listening by category and difficulty

The getListeningsByCategoryAndDifficulty method is a key part of the backend system that supports personalized content delivery in both the Main Session and Practice Session. This function retrieves listening activities that match both a selected category and a specified difficulty level, based on the parameters received from the frontend (categoryId and difficultyLevelId). It queries the Listening collection using Mongoose's find() function to filter entries that belong to the given category and difficulty level simultaneously. The .populate() function is used on the fields QnA, category, and difficultyLevel to include relevant relational data in the final response, ensuring that the frontend receives all necessary context for rendering the listening task and associated questions. If no matches are found, a 404 status is returned with an appropriate message. On successful retrieval, the method returns a 200 status along with the relevant listening activities in JSON format. This function is crucial in the Main Session, where the system automatically recommends tasks based on a student’s current level and chosen topic, and in the Practice Session, where students can manually select both a category and difficulty level to practice more flexibly.

**Retrieve Difficulty Levels**

A black screen with colorful text

AI-generated content may be incorrect.

Figure 17. Get Difficulty Levels

The getDifficultyLevels method plays a crucial role in enabling personalized learning within the Practice Session of the system. This asynchronous function retrieves all available difficulty levels from the DifficultyLevel collection in the MongoDB database using Mongoose's find() method. These difficulty levels help categorize listening activities by complexity—such as beginner, intermediate, or advanced—which allows students to select tasks appropriate to their language proficiency. Upon a successful query, the method responds with a status code 200 and returns the list of difficulty levels in JSON format. If an error occurs during the database operation, a 500 status code is returned along with an error message. This method ensures that the frontend interface can dynamically populate difficulty level options for students, especially in the Practice Session, where users are given the autonomy to explore and practice tasks based on their chosen level of challenge.

**Retrieve Difficulty Levels By Id**

A computer screen shot of a black screen

AI-generated content may be incorrect.

Figure 18. Get Difficulty level by id

The getDifficultyLevelById method is designed to fetch a single difficulty level from the database based on its unique identifier. This method is essential when the application needs to display or process details of a specific difficulty level—such as when editing a level or filtering activities that match a selected level. It uses DifficultyLevel.findById(req.params.id) to locate the document in the DifficultyLevel collection using the provided ID parameter. If no matching difficulty level is found, the method returns a 404 status code with a descriptive message indicating that the resource does not exist. If the difficulty level is successfully found, it is returned to the client with a 200 status code in JSON format. In the event of a database access error, a 500 error response is sent. This method ensures that the system can efficiently retrieve and manage difficulty level data to support dynamic content rendering and logic handling across different user interactions, such as adaptive content delivery and session configuration.

**Retrieve All Questions And Answers**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 19. Get All QnA s

The getAllQnAs method is responsible for retrieving all the Question and Answer (QnA) entries stored in the database. This functionality is essential for the backend to provide the complete set of QnA data, which can be used for rendering questions during listening sessions, displaying them for review, or administering QnA content through an admin interface. The method uses the QnA.find() function, which queries the entire QnA collection and returns an array of all documents. If the query executes successfully, the method responds with an HTTP status code 200 and the list of QnA entries in JSON format. If any error occurs during the process—such as a database connectivity issue—the method catches the error and responds with a 500 status code along with a descriptive error message.