Development of a Multilingual Language Learning Web Application

Trong Phat Nguyen

Haaga-Helia University of Applied Sciences
Bachelor of Business Administration
Business Information Technology
Bachelor Thesis
2025

Abstract

Author(s)
Trong Phat Nguyen

Degree
Bachelor of Business Administration

Thesis Title
Development of a Multilingual Language Learning Web Application

Number of pages and appendix pages
97 + 14

This thesis details the development, implementation, and evaluation of a multilingual language learning web application designed to address the limitations of existing tools, particularly the lack of contextual grammatical information needed for languages like Finnish. The primary objective was to create a user-friendly, AI-powered platform that enhances vocabulary acquisition and retention through context-aware learning.

The application was built using the MERN stack (MongoDB, Express.js, React.js, Node.js) and leverages the OpenAI API for natural language processing tasks such as translation, grammar explanation, and automated flashcard generation. Key features include AI-generated flashcards providing detailed context, a spaced repetition system (SRS) to optimize memory recall based on user performance, integrated gamification elements (XP, levels, streaks, badges) to enhance user motivation and engagement, and personalization features like AI-assisted flashcard refinement and adaptive testing. Development followed an iterative methodology inspired by Agile principles, incorporating continuous testing and refinement.

Evaluation through a user survey (N=32) indicated high overall satisfaction (8.0/10) and usability (8.2/10). The AI-powered flashcards and spaced repetition system were perceived as effective tools for vocabulary learning and retention (average ratings 7.6-7.9/10). Gamification features showed moderate success in engaging users (7.1-7.2/10), while personalization features, though functional, were identified as areas requiring further development.

This project successfully demonstrates the integration of AI, spaced repetition, and gamification to create a comprehensive language learning tool that emphasizes contextual understanding, addressing a notable gap in the current EdTech landscape. While the evaluation shows promising results, the lack of formal statistical significance testing is noted as a limitation.

Key words
Language Learning, Artificial Intelligence (AI), Spaced Repetition System (SRS), Gamification, Educational Technology (EdTech), Natural Language Processing (NLP)
Sources and related content

Table of Contents

1 Introduction

With the rise of self-paced online learning, language learning applications have become an integral part of modern education. Popular applications such as Duolingo and Anki primarily focus on gamification and spaced repetition, respectively. However, most lack an integrated system that combines sentence-based parsing, interactive quizzes, and AI-driven vocabulary correction. This project aims to fill this gap by offering a context-aware flashcard system that provides meaningful sentence-based learning, rather than relying solely on memorizing isolated words.

The focus of this thesis is the development and implementation of a multilingual language learning application that enhances vocabulary acquisition through AI-powered translation, flashcards, and spaced repetition. The motivation for this project arose from personal challenges faced during the process of learning Finnish, where existing translation tools, such as Google Translate and Sanakirja.fi, consistently failed to provide sufficient grammatical context. This led to the creation of an interactive and personalized tool aimed at improving vocabulary retention and comprehension.

The significance of this topic lies in its contribution to digital education and personalized learning technologies. Language learning remains a challenge, particularly for those acquiring complex grammatical structures (Aguion et al. 2021). As technology advances, AI-driven tools have become increasingly relevant in optimizing language acquisition by offering context-based learning and automated review systems (Alhusaiyan, E. 2025). This thesis explores how AI and automation can enhance traditional flashcard-based learning methods, making them more adaptive and efficient.

Additionally, this work aligns with AI-powered education trends, where machine learning and natural language processing (NLP) are used to enhance personalized learning experiences. By leveraging automation, adaptive learning strategies, and AI-enhanced corrections, this application provides a data-driven approach to language learning, improving efficiency and long-term retention.

1.1 Vocabulary

Adaptive Learning Strategies	Adaptive learning is a technique to use data-driven instruction to adjust and tailor learning experiences to meet the individual needs of each student. Adaptive learning systems can track data such as student progress, engagement, and performance, and use the data to provide personalized learning experiences. (Monclair State University)
AI-driven Vocabulary Correction	Using artificial intelligence to identify and correct errors in vocabulary usage
API (Application Programming Interface)	A set of rules and specifications that software applications use to communicate with each other. (Altexsoft Editorial Team 2024) In this context, it allows the frontend to communicate with the backend and to access OpenAI's services.
API prompt	The
Artificial Intelligence (AI)	Artificial intelligence (AI) is a set of technologies that enable computers to perform a variety of advanced functions, including the ability to see, understand and translate spoken and written language, analyze data, make recommendations, and more. (Google Cloud)
Automated Review Systems	Systems that use predefined algorithms to schedule and manage review sessions, optimizing the timing of content review based on factors like retention and learning progress.
Backend	The server-side component of a web application that handles data storage, processing, and API requests.
Backup database	A secondary database used to store copies of data, enabling recovery in case of data loss.
CI/CD (Continuous Integration/Continuous Development)	A set of practices that automate the building, testing, and deployment of software, ensuring rapid and reliable releases.
Codebase	The collection of source code used to build a software application.
Code refactoring	The process of restricting existing computer code without changing its external behavior.
Context-Aware Learning	Learning that considers the surrounding information (e.g., sentence structure) to provide a deeper understanding.
Cookies (in web development)	Small pieces of data stored on a user's computer by a web browser, used to remember information about the user (e.g., login status)
CSS	Cascading Style Sheets; a style sheet language used for describing the presentation of a document written in HTML or XML.

Cypress	A JavaScript end-to-end testing framework for web applications.
Custom hooks (in React)	Reusable functions in React that allow developers to add stateful logic to components.
Database Management	The process of storing, organizing, and retrieving data efficiently, in this context, using MongoDB.
Database Query	A request for data from a database.
Data persistence	The ability of data to survive after the process that created it has ended.
Data-Driven Approach	A method that relies on data analysis to inform decision-making (Mucci, T. 2024)
Digital Education	Education that utilizes technology (Wikipedia)
EdTech	Educational technology; technology used to support teaching and learning
Encryption	The process of encoding information to make it unreadable without a key, used to protect sensitive data.
Environment variables	Variables set outside the program, typically through the operating system, used to store sensitive information like API keys.
Express.JS	A fast, minimalist web framework for Node.JS, used for building the backend API. (Express.JS 2024)
Feature branch	A branch it Git used to develop a specific feature in isolation.
Figma	A web-based design tool used for creating user interface designs.
Flashcard	A card with information on both sides, used for memorization
Fly.io	A platform used for deploying and scaling applications.
Form submissions	The act of a user sending data entered into a web form to the server.
Free tier	A level of service for a product (like a database) that is provided without charge, but often with limitations.
Frontend	The client-side component of a web application that provides user interfaces.
Full-stack development	The process of developing both the frontend (user interface) and the backend (server-side logic) of a web application.

Gamification	The application of game-design elements in non-game contexts. (Buljan, M. 2021)
GDPR	General Data Protection Regulation; a regulation in EU law on data protection and privacy in the EU and the European Economic Area (EEA). (Wikipedia, 2025)
Git workflow	A set of practices and conventions for using Git to manage code changes.
Grammatical Context	The structural rules and relationships within a sentence or phrase that affect the meaning of words.
JSON Web Tokens (JWT)	A standard for securely transmitting information between parties as a JSON object, used for user authentication and session management (GeeksforGeeks 2025)
Kanban board	A visual workflow management tool used to track tasks and progress.
Lazy loading	A technique that defers loading of resources until they are needed, improving initial page load time.
Lint checks	Automated checks for code quality and style.
LocalStorage	A web storage technology that allows web applications to store data locally within the user's browser.
Machine Learning (ML)	A type of AI that allows computers to learn from data without being explicitly programmed (Brown, S. 2021)
Mock data	Sample data used for testing software before the actual data is available.
Modularizing components	The process of restructuring existing computer code without changing its external behavior.
Mongoose	A MongoDB object modeling tool designed to work in an asynchronous environment (Hall, S. 2024)
Multilingual	Involving or using multiple languages.
MVC (Model-View-Controller)	An architectural pattern that separates an application into three interconnected parts: the Model (data), the View (user interface), and the Controller (logic) (Codeacademy Team s.a.)
Natural Language Processing (NLP)	A branch of AI that deals with the interaction between computers and human (natural) languages (Holdsworth & Stryker 2024)
Node.JS	A JavaScript runtime environment that allows developers to build server-side applications. (Node.JS 2025)

OpenAI	An artificial intelligence research and deployment company. (OpenAI Platform 2025). In this application, OpenAI's NLP models are used for tasks such as translation and vocabulary correction.
OpenAI's NLP models	Artificial intelligence models developed by OpenAI, used within this application for language-based task (OpenAI Platform 2025)
Personal data leaks / Data breaches	Incidents where sensitive user information is accessed or disclosed without authorization
Personalized Learning	An educational approach that tailors learning experiences to individual student needs and preferences.
Privacy measures	Actions and policies implemented to protect user data and maintain confidentiality.
Pull request	A request to merge changes from one Git branch into another.
React.JS	A JavaScript library for building user interfaces, used for the frontend of the application. (React s.a.)
React Testing Library	A library for testing React components.
Responsiveness (in web development)	The ability of a website or application to adapt to different screen sizes and devices.
Schema	The structure or format in which data is organized and stored
Sentence-Based Parsing	The process of analyzing the grammatical structure of a sentence.
Spaced Repetition	A learning technique that involves reviewing material at increasing intervals to optimize retention (Kwantlen Polytechnic University s.a.)
Spaced Repetition System (SRS)	A Spaced Repetition System, commonly referred to as SRS, is a system for efficiently reviewing information. It uses a statistical algorithm to model the optimal times a piece of information should be seen for a learner to remember it.
Supertest	A library for testing HTTP assertions.
Test coverage	A measure of how much of the code is executed by tests.
UI glitch	A minor error or malfunction in the user interface, causing visual or functional problems.
Utility functions	Functions that perform common tasks and are used throughout a codebase.
Vitest	A fast unit test framework powered by Vite.

Vocabulary Acquisition	The process of learning and memorizing new words
Vocabulary Retention	The ability to remember learned words over time.

1.2 Objectives and Scope of the Thesis

The primary objective of this thesis is to develop, refine, and evaluate an AI-powered language application. The key goals include:

- Developing a fully functional and user-friendly application that allows learners to study vocabulary through AI-enhanced flashcards.
- Enhancing AI-powered translation and vocabulary correction features to improve learning accuracy.
- Implementing a spaced repetition system to reinforce long-term vocabulary retention.
- Testing usability and performance through user feedback and technical evaluation.
- Evaluating the learning impact of AI-enhanced study methods based on user engagement and progress tracking.

The scope of this thesis includes the design, implementation, and assessment of the application. The project does not focus on mobile application development (only a web-based platform) or large-scale user testing, as the testing will be limited to a small group. Additionally, the project lies on OpenAI's API rather than developing a custom NLP model.

1.3 Research Questions / Development Tasks

To achieve these objectives, the thesis will address the following key research questions:

- How effective is AI-powered vocabulary correction and translation in improving language retention compared to traditional flashcard systems?
- What are the primary technical challenges in developing a scalable and efficient AI-powered language learning platform?
- How does spaced repetition, when combined with AI-enhanced features, impact vocabulary retention and user engagement?
- What are the advantages and limitations of using NLP and AI in language learning applications?
- How do users perceive the effectiveness and usability of the application, and how can it be improved?

The development tasks include:

- Implementing and refining AI-powered flashcards, translations, and vocabulary corrections.
- Enhancing the spaced repetition system to optimize learning schedules.
- Improving the user interface and experience for effective engagement.
- Conducting testing and data collection to access application performance.
- Analyzing user feedback and refining the application accordingly.

The development tasks are directly tied to the research questions and objectives, ensuring that:

- Each feature addresses a key research question (e.g., AI-powered flashcards for vocabulary retention)
- Technical challenges are tackled through systematic implementation and testing.
- User experience is improved based on measurable feedback and refinements.

1.4 Key Concepts

Several key concepts underpin the theoretical framework and development of this project:

- AI-Powered Language Learning – Utilizing NLP and machine learning models to enhance vocabulary acquisition.
- Spaced Repetition Algorithm (SRS) – A learning technique that schedules review sessions at increasing intervals to optimize memory retention
- Natural Language Processing (NLP) – AI-driven methods that enable machines to process and understand human language, such as translation, grammar correction, and contextual analysis.
- User Experience (UX) in Educational Technology – Studying how learners interact with digital tools and how usability impacts engagement and learning outcomes.
- Learning Analytics – Measuring user engagement, vocabulary retention rates, and AI accuracy to assess the effectiveness of the application

2 Theoretical Framework

This chapter establishes the theoretical foundations underpinning the development of a multilingual language learning application. It explores key concepts and technologies shaping modern language learning, including the evolution of language learning technologies, the principles of spaced repetition and gamification, and the role of artificial intelligence (AI) in enhancing flashcards and personalization. These topics are examined through a review of existing literature, providing a conceptual framework for understanding how such approaches optimize vocabulary acquisition and retention. The discussion draws on established theories and empirical studies to contextualize the project's objectives, setting the stage for their practical application in later chapters.

2.1 Overview of Language Learning Technologies

Language learning technologies have evolved significantly, leveraging advancements in artificial intelligence (AI), gamification, and adaptive learning (Alhusaiyan, 2025). Traditional methods such as textbooks and rote memorization have been supplemented or replaced by digital platforms that provide interactive learning experiences.

Modern AI-powered applications utilize Natural Language Processing (NLP), machine learning models, and spaced repetition algorithms to enhance learning efficiency.

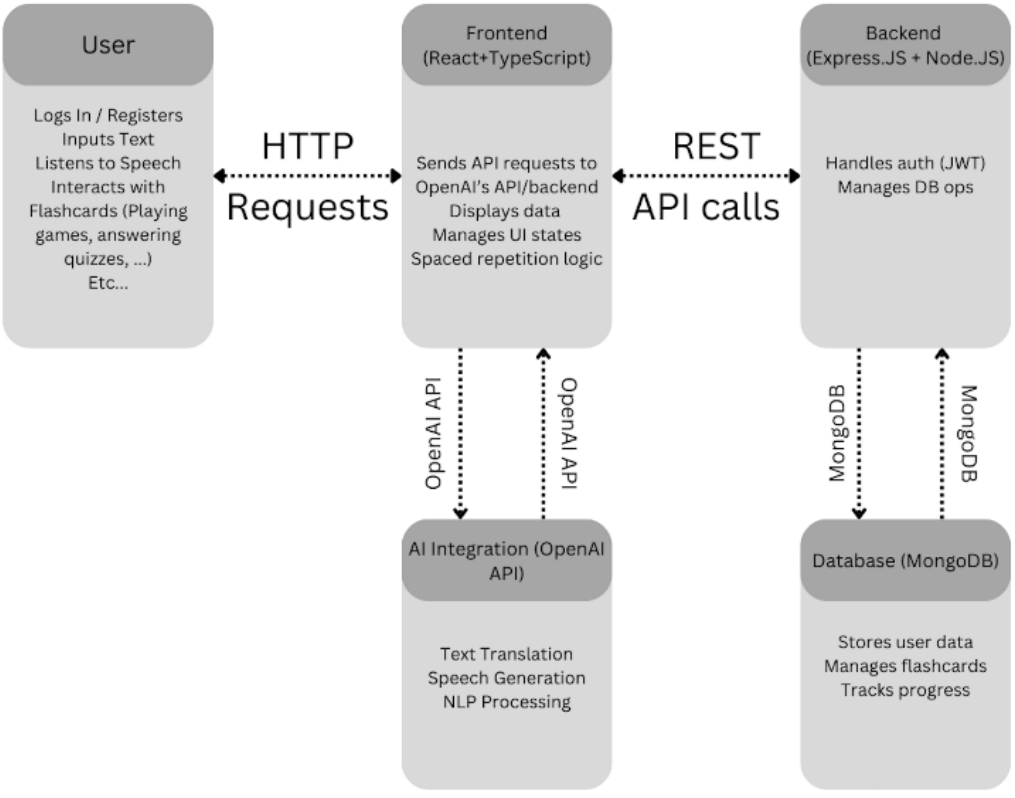


Figure 1: Language Learning App Architecture diagram illustrates user interaction, frontend (React/TypeScript), backend (Express/Node.js), API communication (REST, OpenAI), and database (MongoDB) components.

2.2 Spaced Repetition and Gamification in Learning

To provide a foundation for the application's learning mechanisms, this section delves into the principles of spaced repetition and gamification. These methodologies are crucial for optimizing vocabulary acquisition and learner engagement, and their theoretical basis is discussed in the following subsections.

2.2.1 Spaced Repetition

Spaced repetition is a scientifically backed learning technique designed to enhance long-term memory retention. It involves reviewing information at gradually increasing intervals, based on the user's past performances. This technique is grounded in Ebbinghaus' forgetting curve, which suggests that people forget learned material rapidly unless it is reinforced at strategic time intervals (Kwantlen Polytechnic University s.a.)

Numerous studies have shown that spaced repetition is highly effective for vocabulary acquisition and long-term retention. Research by Cepeda et al. (2006) demonstrated that spacing out learning sessions significantly improves memory compared to massed learning (cramming). Additionally, Kang (2016) highlights that spaced repetition enhances active recall, making it preferred method in educational technology.

Many language learning applications implement spaced repetition to improve learning efficiency. For instance, Anki bases its algorithm on the SuperMemo2 (SM-2) system with several modifications, using "ease factors" to adjust the intervals between reviews. After reviewing a card, users rate how well they remembered it (Again, Hard, Good, Easy), and this rating determines when the card will reappear. Anki also incorporates the concepts of "stability," referring to how long a memory lasts, and "retrievability," referring to the probability of recalling a memory at a given time, into its calculations. More recently, Anki introduced a newer algorithm called FSRS, which aims to model memory states even more accurately (Anki FAQ).

Similarly, Duolingo applies a machine learning model known as "half-life regression" (HLR) to predict how quickly users are likely to forget a word or concept. This model estimates the "half-life" of knowledge within a user's memory and schedules reviews accordingly, while also adjusting lesson frequency based on past mistakes and demonstrated strengths (Blanco, C. 2024).

Memrise follows a comparable approach by predicting when users are likely to forget a word or sentence and scheduling reviews based on these estimates. Correct answers move the item to progressively longer intervals, following a standard schedule of 4 hours, 12 hours, 24 hours, 6 days, 12 days, 48 days, 96 days, and 6 months. In contrast, incorrect answers reset the item to the earliest interval, ensuring that weaker memories receive more frequent reinforcement (Memrise 2024).

2.2.2 Gamification

Gamification refers to the application of game-design elements in non-game contexts to enhance user engagement, motivation, and learning outcomes (Buljan, 2021). In the context of this user language learning application, gamification is integrated to complement the AI-powered features and spaced repetition system, making vocabulary acquisition more interactive and rewarding. This section outlines the gamification mechanics implemented in the application, their theoretical grounding, and their alignment with the project's objectives of improving user engagement and long-term retention.

Gamification in educational technology leverages psychological principles such as intrinsic and extrinsic motivation, as well as the concept of flow (Csikszentmihalyi, 1990). By incorporating rewards, progression systems, and challenges, gamification encourages users to remain committed to their learning goals. Hamari et al. (2014) conducted a broad review of empirical studies on gamification across various domains, including education. This review highlighted that gamification, when implemented thoughtfully, can lead to positive outcomes such as increased user activity, engagement, and perceived learning. Specifically, the review found that common gamification elements like points, badges, and leaderboards were frequently used in educational settings and often associated with increased student participation. However, the authors also emphasized that the effectiveness of gamification is highly context-dependent and reliant on careful design.

While gamification offers numerous benefits in educational settings, it is important to also consider potential drawbacks. For instance, an overreliance on extrinsic rewards such as points and badges may undermine intrinsic motivation, reducing learners' engagement for the sake of learning itself. Additionally, if the game mechanics are not closely aligned with the learning objectives, gamification can lead to superficial engagement, where learners focus on "gaming the system" rather than achieving meaningful understanding. The effectiveness of gamification can also vary significantly across individuals; while some learners may find it highly motivating, others might perceive it as distracting or even demotivating. Furthermore, designing effective gamified learning experiences requires careful attention to the alignment of game mechanics, learning goals, and user needs, as poorly executed designs can ultimately be ineffective or counterproductive.

In this application, efforts have been made to mitigate these challenges by integrating gamification with meaningful learning tasks, providing personalized challenges, and offering a balance of intrinsic and extrinsic rewards. For instance, the application focuses on providing immediate feedback on correct answers and progress visualization to foster a sense of competence, which is a key component of intrinsic motivation.

The design of the gamification system draws inspiration from successful language learning platforms like Duolingo, which uses daily streaks and experience points (XP), and Memrise, which combines multimedia rewards with spaced repetition (Blanco, 2024; Memrise, 2024). However, this project extends beyond these models by integrating context-aware learning and AI-driven corrections into the gamified experience, addressing the gap in existing tools identified in the thesis introduction."

2.3 AI-Powered Flashcards and Personalization

Artificial intelligence plays a significant role in several aspects of the application. This section outlines the AI-powered features implemented to provide personalized and efficient learning tools, with a focus on flashcard generation and grammar analysis.

2.3.1 AI-Powered Flashcards

Artificial intelligence (AI) enhances language learning by automating content generation and personalizing educational experiences (Alhusaiyan, 2025). Natural Language Processing (NLP), a branch of AI, enables systems to process and analyze human language, supporting tasks such as translation and contextual interpretation (Holdsworth & Stryker, 2024). In the context of flashcards, AI can transform static memorization tools into dynamic, context-aware aids that provide translations and examples, improving comprehension over traditional methods (Kang, 2016). This automation reduces cognitive overload, allowing learners to focus on understanding rather than manual preparation (Alhusaiyan, 2025). Personalization further tailors learning to individual needs, leveraging analytics and adaptive techniques to align content with learner proficiency (Monclair State University, n.d.). Research indicates that such personalized approaches enhance engagement and vocabulary retention by ensuring meaningful context (Cepeda et al., 2006; Kang, 2016).

2.3.2 Personalization

Personalization further enhances learning by adapting educational content to individual user needs, leveraging performance analytics and adaptive techniques (Monclair State University, n.d.). This approach can include tailoring difficulty levels, customize learning paths, and provide targeted feedback, often inspired by diverse educational assessment models (Nguyen, 2019). Research indicates that personalized learning improves engagement and vocabulary retention by aligning content with learner proficiency and ensuring meaningful context (Cepeda et al., 2006; Kang, 2016).

3 Description of the Product-Based Thesis

This chapter provides a detailed overview of the developed multilingual language learning application. It covers the application's target users, key features, and development progress, highlighting the design choices and technical considerations involved in its creation.

3.1 Target Audience of the Application

The multilingual language learning application developed in this thesis targets a diverse yet specific audience, each with distinct needs and interests that align with the project's objectives. The primary audience includes three key groups: language learners seeking advanced tools for vocabulary retention, educational technology researchers exploring AI-driven learning solutions, and developers interested in integrating AI with full-stack development in educational technology (EdTech) applications.

The first group, language learners, encompasses individuals engaged in self-directed or supplementary language acquisition, particularly those tackling languages with complex grammatical structures, such as Finnish. These users often require more than traditional flashcard methods or generic translation tools, which may lack sufficient contextual depth (Aguion et al., 2021). The application addresses this need by offering an AI-powered platform that enhances vocabulary retention through features like context-aware flashcards and spaced repetition, catering to learners who value personalization and efficiency in their study process.

The second group consists of educational technology researchers who investigate the intersection of AI and language learning. This audience is interested in how artificial intelligence, including Natural Language Processing (NLP) and adaptive learning algorithms, can optimize educational outcomes (Alhusaiyan, 2025). The application serves as a practical case study, demonstrating the potential of AI-driven solutions to improve vocabulary acquisition and user engagement, thus contributing to ongoing scholarly discussions in the field.

The third group includes developers and technologies exploring the application of AI within full-stack development for EdTech purposes. This audience seeks insights into the technical implementation of tools like OpenAI's NLP models, React.JS frontends, and Node.JS backends, as well as the integration of gamification and personalization features. The application offers a model for building scalable, user-centric learning platforms, appealing to those interested in both the theoretical and practical aspects of software development in education.

By targeting these groups, the application bridges practical utility for learners with research and development value for academics and technologists, reflecting the thesis's dual focus on product creation and scholarly contribution.

3.2 Features and Functionalities of the Application

The language learning application developed in this thesis is designed to enhance vocabulary acquisition through an AI-powered, user-centric platform. It integrates advanced technologies and strategies to address the needs of language learners, educational technology researchers, and developers, as outlined in Section 3.1. The core functionalities include automated translation and flashcard generation, spaced repetition for optimized retention, interactive learning sessions, and personalized tailoring options, supported by a responsive design and user management features. These capabilities reflect the application's goal of making language learning more accessible, efficient, and contextually rich, inspired by challenges encountered in learning languages like Finnish, where existing tools often lack grammatical depth.

A primary feature is the ability to input sentences or paragraphs, which the application automatically translates and parses into individual words using AI-driven Natural Language Processing (NLP). Users can save these words as flashcards to personalized decks, categorized by language and tags, streamlining the process of vocabulary creation. This automation reduces the manual effort required for traditional flashcard preparation, addressing a common barrier for learners seeking contextual understanding. The system further enhances comprehension by providing translations, pronunciations, explanations, and contextual examples for each card, ensuring vocabulary is learned within meaningful frameworks.

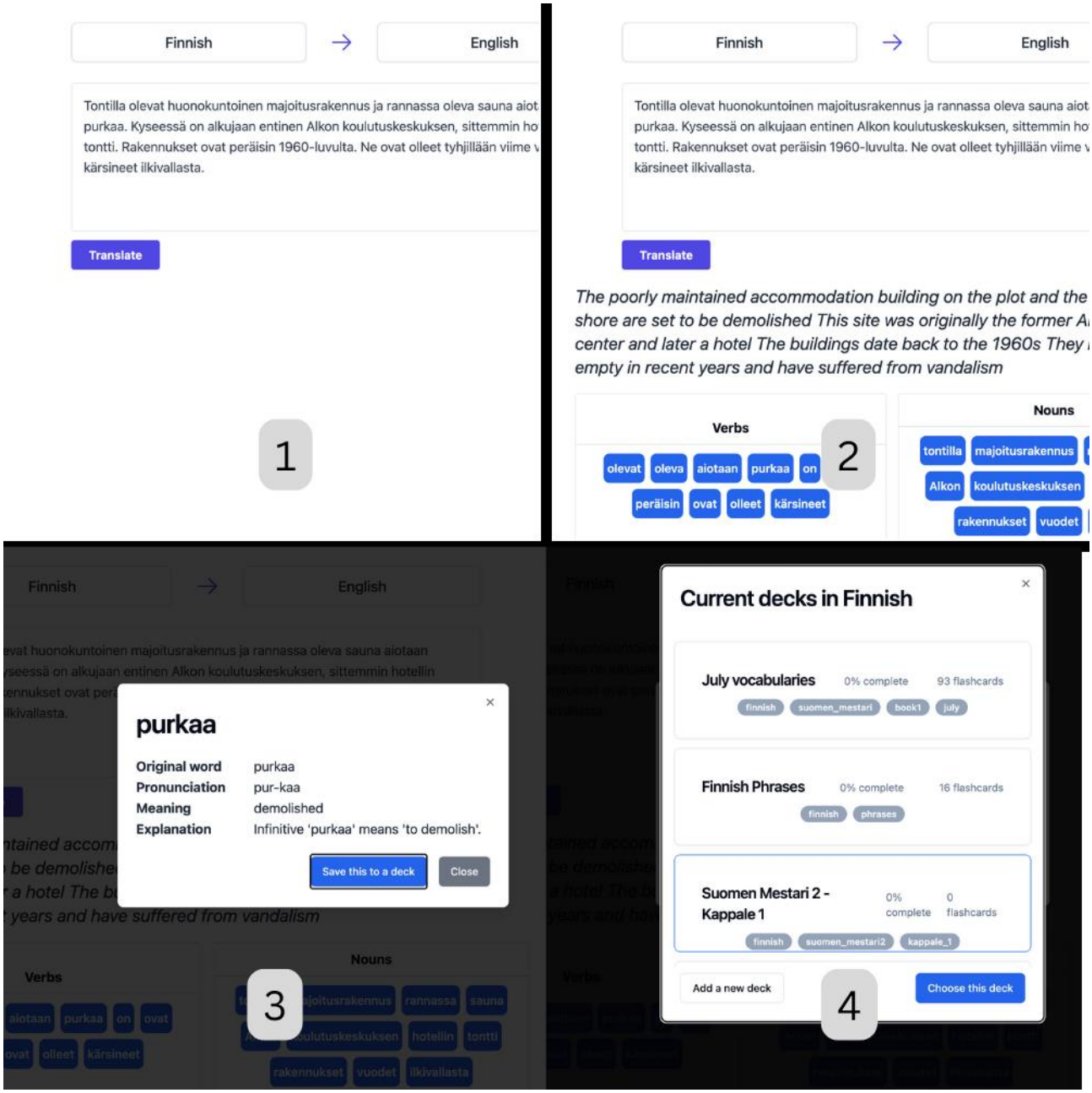


Figure 2: Workflow for AI-powered flashcard creation: (Top-left, 1) User inputs sentence into translation box, (Top-right, 2) Application displays translation and parsed words, (Bottom-left, 3) Individual words presented as interactive flashcards with details, (Bottom-right, 4) User saves selected flashcards to a chosen deck

To optimize long-term retention, the application employs a spaced repetition algorithm, dividing flashcards into three states - Not Studied, Learning, and Completed - based on user performance. Interactive learning sessions, accessible via a dedicated Learning Page, present quiz-based questions with customizable options, such as the number and type of flashcards or shuffling for variety. Performance metrics, including correctness and response time, inform the algorithm's scheduling of review dates, aligning with evidence-based learning techniques to reinforce memory efficiently.

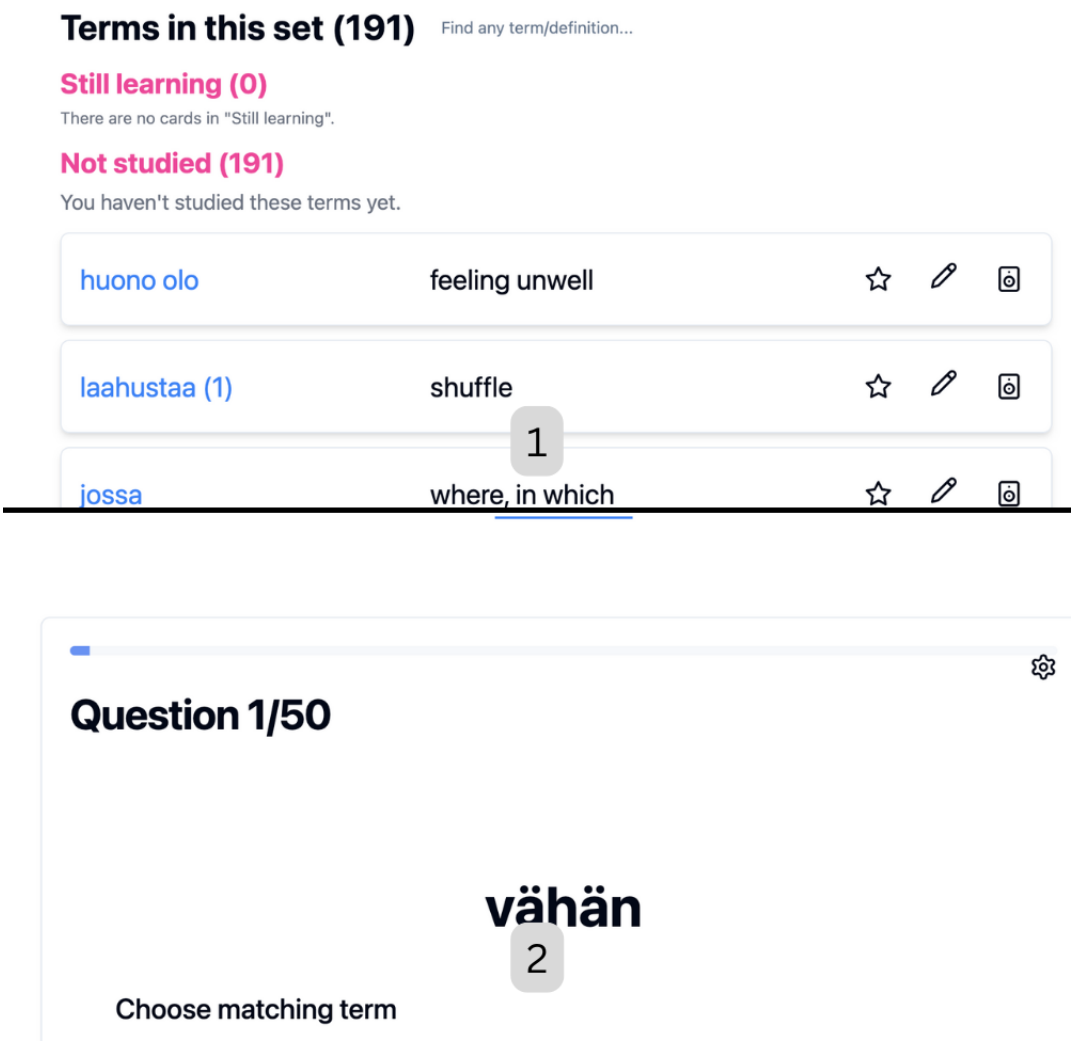


Figure 3: Spaced Repetition System interface: (Top, 1) Deck Details view showing vocabulary categorized by learning state ('Not Studied', 'Learning', 'Completed'), (Bottom, 2) Interactive Learning Page presenting a quiz question for the selected deck.

Personalization is a key strength, illustrated by the tailoring feature on the Deck Details page. Recognizing that AI-generated translations from sentences can sometimes be vague or inaccurate due to contextual nuances, this feature allows users to refine translations by selecting from AI-revised options or web-scraped alternatives from resources like Sanakirja.fi (for Finnish). Additional personalization includes adaptive testing formats - such as fill-in-the-blank passages, word scrambling, and synonym matching - designed to reinforce vocabulary through varied engagement methods. Users can also modify decks, mark favorites, and track learning history, fostering a tailored experience that adapts to individual progress.

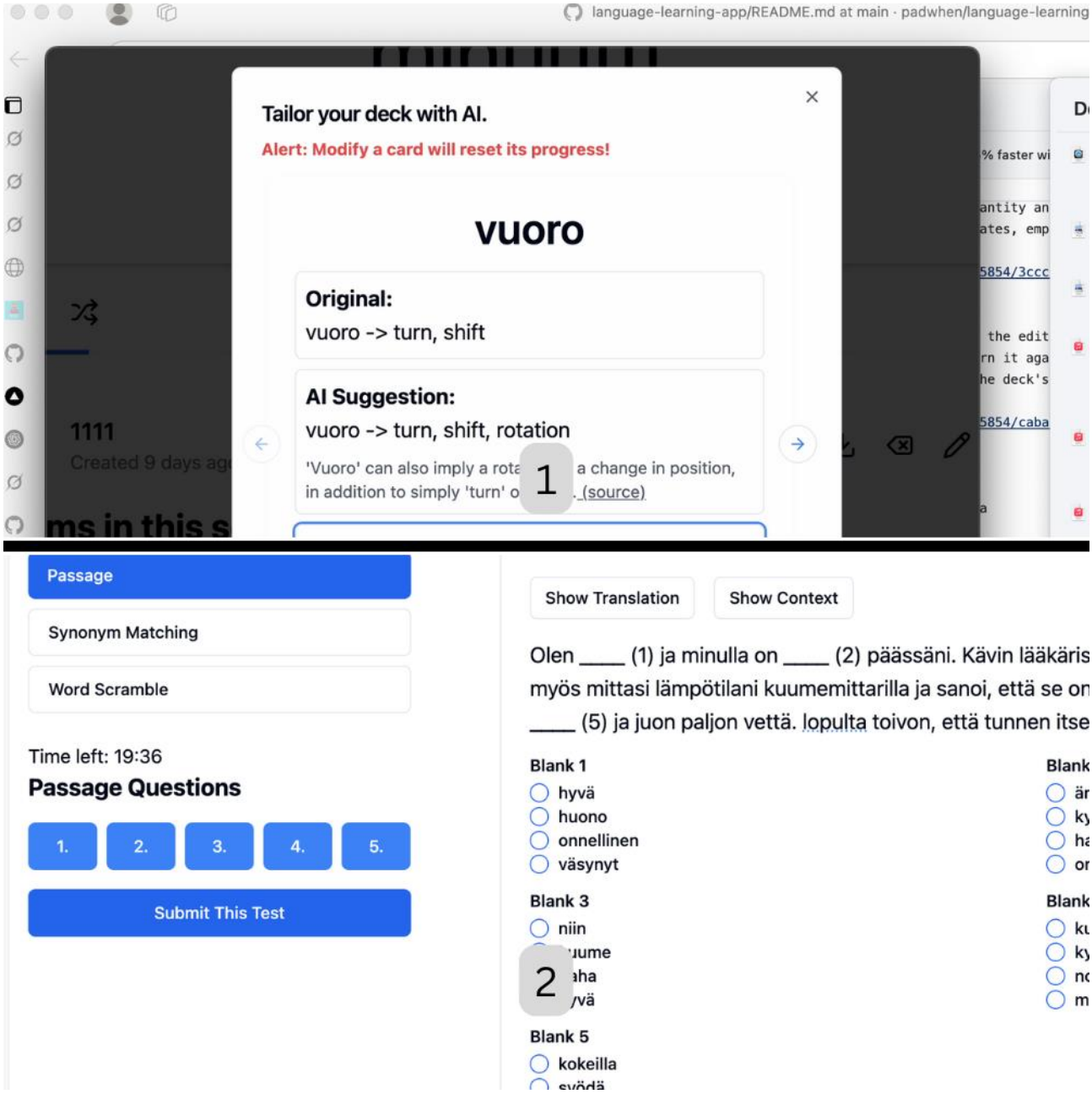


Figure 4: Personalization features: (Top, 1) AI-Tailored Flashcards interface allowing users to review and refine AI-generated translations against dictionary suggestions, (Bottom, 2) Adaptive Testing Page offering varied quiz formats like fill-in-the-blank passages.

Supporting these core functionalities are essential user management features, including login and registration for secure access, profile editing (e.g., avatar, username updates), and deck creation with customizable attributes (name, languages, tags). The application's responsive design ensures seamless usability across desktop and mobile devices, enabling on-the-go learning – a critical consideration for modern leaners. While gamification elements like streaks and progress tracking are in development, the current feature set prioritizes vocabulary retention and contextual learning, with plans for future enhancements like game-based interactions and advanced testing modes (see Section 7.2).

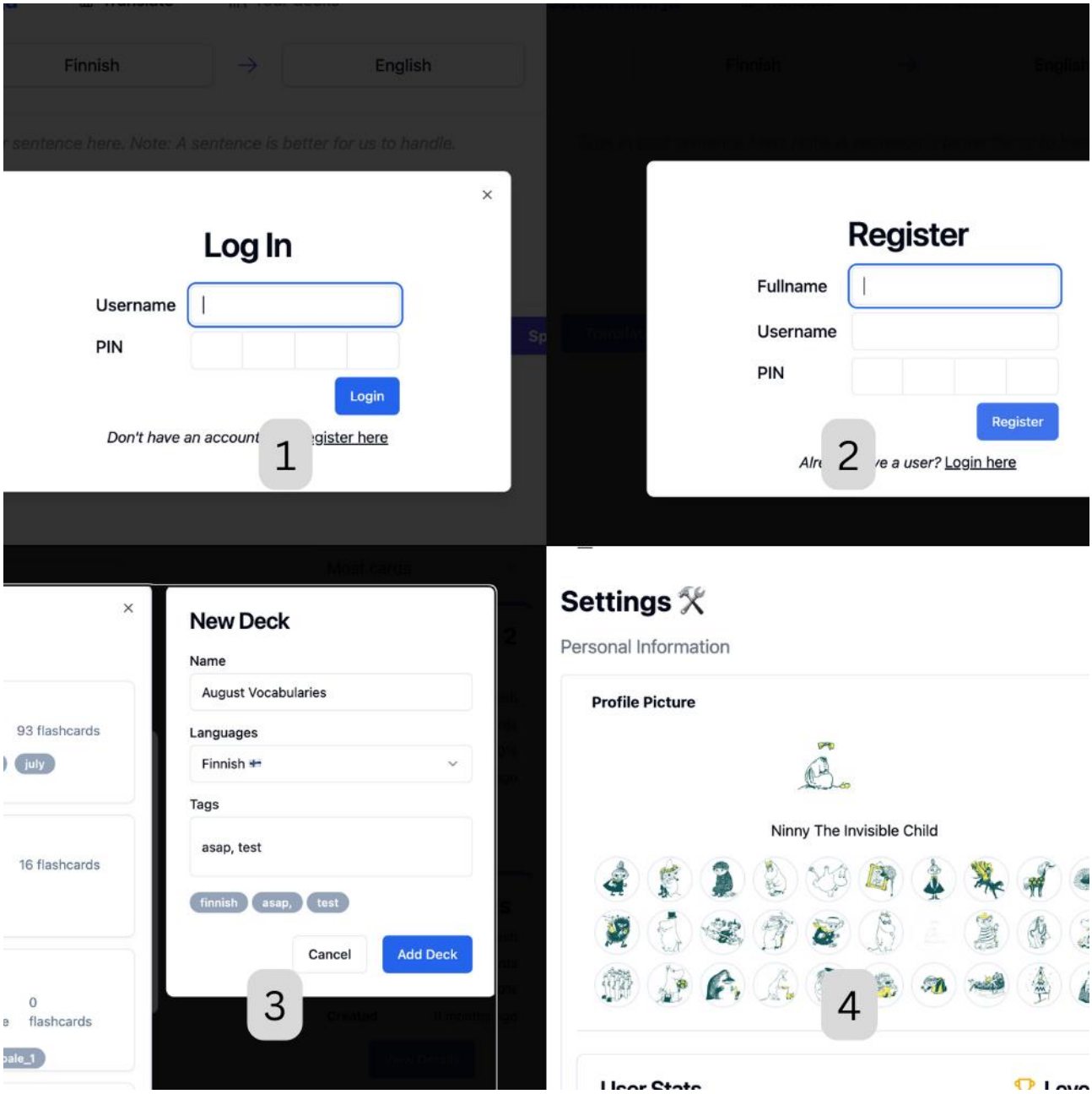


Figure 5: Key user management interfaces: (Top-left, 1) Login form, (Top-right, 2) User registration form, (Bottom-left, 3) Modal for creating a new vocabulary deck, (Bottom-right, 4) User settings page for profile customization.

Together, these features create a cohesive platform that not only supports language learners in mastering vocabulary but also serves as a practical demonstration of AI integration and full-stack development for researchers and technologists. The application's design reflects a response to real-world learning challenges, offering a structured, personalized study plan that enhances both retention and engagement.

3.3 Development Methodology and Tools

The development of the language learning application followed a flexible and iterative process, focusing on solving real challenges in learning Finnish. This section explains how the application was designed and built, covering the tools, development stages, and improvements made over time. Inspired by Agile principles, the approach emphasized adaptability and user needs. The process involved planning, using modern technologies, and continuously refining the application based on feedback.

The following sections describe the key stages of development, including feature enhancements in gamification and adaptive learning, as well as efforts to improve the application's stability and quality. The application was built using React.JS, Node.JS, MongoDB, and OpenAI's API. Task management was handled with GitHub's Kanban board, and GitHub Actions was used for continuous integration. This approach balanced structured planning with organic growth, as the developer played both the role of creator and main user.

3.3.1 Key Development Stages

The development of this application started as a personal challenge—manually extracting and studying Finnish vocabulary using tools like Google Translate and ChatGPT. This process was time-consuming and lacked grammatical context. Writing down words, meanings, pronunciations, and example sentences in a notebook felt inefficient. To solve this problem, the idea for an automated tool was born. The initial goal was to create an application that could automatically translate text using OpenAI's API, extract vocabulary with detailed metadata, and generate flashcards. These ideas were first outlined in handwritten notes to define the application's scope.

The development process followed an Agile-inspired approach, involving continuous design, development, and refinement. The first step was to build the application's structure: a frontend using React.JS for an interactive interface and a backend with Node.JS and MongoDB to manage data. A simple sketch of the main page was created, featuring a translation input box, detailed output, user information, and flashcard decks. This design helped guide the initial prototype, which used mock data to test functionality before integrating live API calls.

To manage tasks, a quadrant-based prioritization system was used on a GitHub Kanban board. Tasks were sorted by urgency and importance. For example, implementing the translation feature was a top priority since it was central to the app, while UI improvements were important but less urgent. Each task was broken down into smaller coding steps, such as setting up API endpoints or designing flashcard components, to ensure steady progress.

At first, there were no fixed phases or milestones, leading to an unstructured workflow. This eventually caused a pause in development due to a lack of technical knowledge. During this break, additional coursework and small coding projects helped build the necessary skills. With this new experience, development resumed with a clearer plan, following structured phases: prototyping, core feature integration, and iterative improvements. Self-imposed milestones, like launching the translation feature by June 2024, helped maintain focus. This journey reflects how the development process evolved from an unstructured, exploratory approach into a more systematic, Agile-like method that adapted to both technical challenges and personal learning growth.

3.3.2 Feature Enhancements (Gamification, Adaptive Learning)

Feature enhancements, especially gamification and adaptive learning, were shaped by the developer's experience as both the creator and primary user. Initially, the application focused on

translation and flashcard creation on the index page. However, as usage increased, new needs emerged – such as organizing flashcards systematically and studying them interactively. In response, the Deck Details page was introduced first to structure and display flashcards, followed by the Learning Page the following month, which provided a quiz-based study feature. These additions aimed to create a more engaging and structured learning experience.

Adaptive learning features, like deck customization and quiz settings, developed from practical challenges and research. The deck customization feature, implemented in June 2024, allowed users to refine AI-generated translations, addressing inaccuracies in sentence-based outputs – an issue the developer personally encountered while learning Finnish. Quiz customization on the Learning Page enabled users to adjust the number and type of flashcards based on their needs. This feature was informed by research on spaced repetition from sources like GitHub, Stack Overflow, YouTube, and academic papers, ensuring that review intervals and difficulty levels supported memory retention. Development began with handwritten sketches or Figma designs, followed by mock data (e.g., *mockdata.json* files) or hardcoded UI elements, which were gradually refined into fully functional components.

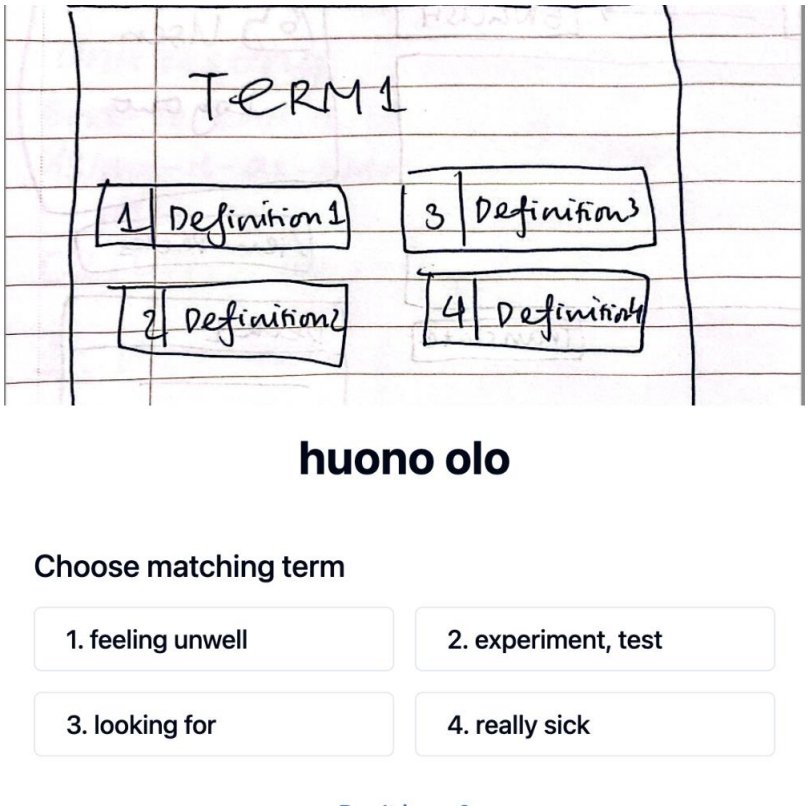


Figure 6: The evolution of the Learning Page interface, from initial hand-drawn sketch (top) to implemented application screen (bottom)

Gamification elements, such as streaks and toast notifications, were introduced later (ongoing as of April 2025) to make learning more engaging. These features were added to enhance motivation as the app continued to grow. However, as the codebase became more complex, new challenges emerged. Every new feature had to be carefully evaluated to ensure it fit within the application's core purpose. Key questions included whether a feature complemented or disrupted existing functionality, whether it played a primary or supporting role, and whether it aligned with the original vision. This reflective approach ensured that enhancements contributed to a more effective and

personalized language learning experience. Future additions, such as game-based interactions, are planned for later iterations (see Section 7.2).

3.3.3 Bug Fixes and Performance Optimization

During development, several bugs and technical challenges emerged. One major issue involved OpenAI's API, which occasionally returned data in inconsistent formats. To address this, a structured schema was added to the API prompt to ensure uniform responses. UI glitches also posed a challenge—early implementations used hardcoded element sizes, preventing the app from adapting properly to different screen sizes, leading to display issues on both large monitors and mobile devices. Additionally, database-related problems arose, such as slow queries and data persistence errors.

Bugs were identified through a combination of testing, trial and error, and extensive debugging with console.log statements. UI inconsistencies were discovered by switching between devices and adjusting CSS for better responsiveness. Database issues were traced by logging query results and resolving connection errors. To improve performance and maintainability, the codebase was refactored by modularizing components, exporting types, organizing utility functions, introducing custom hooks, and implementing lazy loading to optimize page load times. For example, breaking a large file into smaller custom hooks simplified debugging and improved code organization.

Breaking down complex issues into smaller tasks proved to be an effective problem-solving strategy. When errors occurred, reverting to the last working version made troubleshooting more manageable. GitHub Actions and CI/CD pipelines played a crucial role by running automated tests and lint checks before deployment. If a test failed or a linting issue was detected, notifications alerted the developer, allowing errors to be addressed early in the process. Deploying to the live server sometimes revealed additional issues, such as slow API responses that were not apparent during local testing. These improvements enhanced the application's stability and performance, resulting in a smoother user experience.

3.3.4 Code Quality and Testing Improvements

To ensure the application remained reliable and maintainable, both testing and code quality were continuously improved. Testing was conducted with every new feature. For UI validation, Cypress was used to check layout and interactions. Utility functions were tested using React Testing Library or Vitest to verify their correctness. Backend API endpoints were tested with Supertest to ensure data was returned properly and error handling was effective. Custom hooks were tested based on complexity – hooks closely tied to a single component were verified through component tests, while reusable or more complex hooks had dedicated tests to check loading states, state updates, and handling of multiple requests.

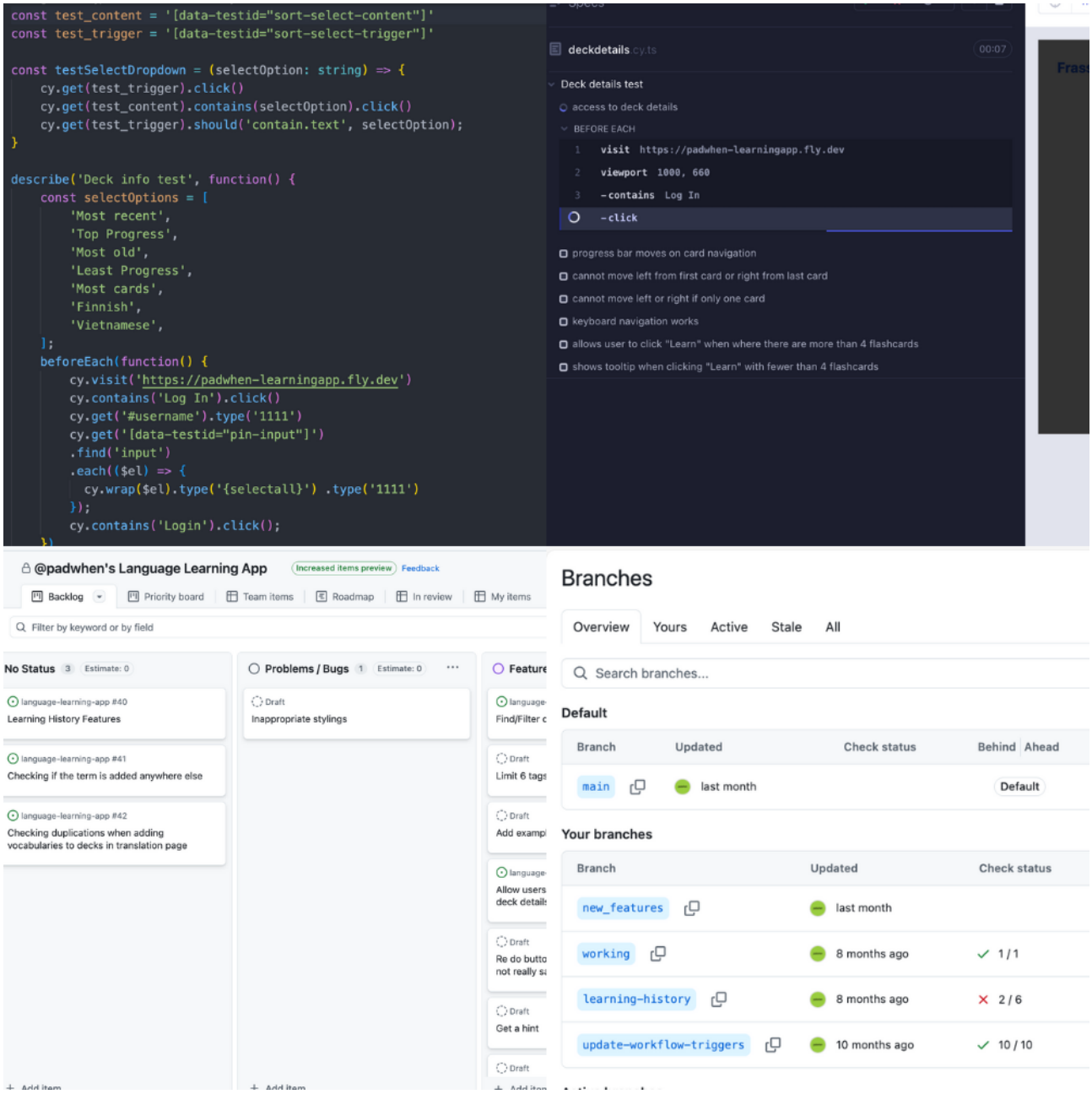


Figure 7: Visual representation of the development lifecycle: (Top left) Example of end-to-end tests ensuring code reliability, (Top right) Cypress browser view executing UI tests, (Bottom left) GitHub Kanban board for visualizing task progress and bug resolution

The testing process followed a structured approach, covering different scenarios:

- Happy cases: Standard usage with valid inputs and expected outputs.
- Error cases: Invalid inputs, such as null values or incorrect formats.
- Edge cases: Unusual inputs, such as extremely large numbers or empty lists.
- Special cases: Unique cases, like handling special characters or whitespace

While the primary focus was on happy cases to maintain core functionality, edge cases were more challenging to test comprehensively. Additionally, real-world usage helped uncover unexpected

issues. When bugs were encountered, they were documented on the GitHub Kanban board for future fixes.

Code quality improved over time through better organization and best practices. The project eventually incorporated over 30 custom hooks, reusable utility functions (as of April 2025), and structured Git workflows, including feature branches and pull requests. Initially, utility functions, hooks, and type definitions were loosely grouped into broad directories like “utils/” and “types/”. However, as the project evolved, related functions and types were moved into specific component folders. For example, the Deck Details page received its own *useDeckDetails* hook and type definitions, making the codebase more modular and easier to maintain. GitHub Actions was used to enforce quality checks, running tests and linting to catch issues early, while the Kanban board helped track tasks and bugs systematically.

Throughout development, challenges frequently rose. Some new features overlapped with existing functionality, requiring careful decisions – integrate them and risk complexity, or create separate implementations and risk code duplication. This was managed through an iterative approach, developing step by step and making adjustments as needed. Testing played a crucial role in identifying bugs, but real-world usage often revealed additional issues, making both strategies essential. Over time, test coverage expanded, and refactoring improved the overall structure, enhancing the application’s stability and maintainability.

4 Data Management and Ethical Considerations

In this chapter, the focus is on how the language learning application handles user data and considers ethical responsibilities. It explains the data collected and how it is managed, the steps taken to protect user privacy, and the methods for storing, securing, and disposing of data. This ensures the app supports personalized learning while keeping user information safe and used responsibly. The sections below cover data collection and handling, ethical considerations, and data storage, security, and disposal, reflecting the app's practical design and ongoing development.

4.1 Data Collection and Handling

The application collects several types of data to support its features and improve the user experience. This includes user account details such as usernames, avatars, XP points, streaks, and achievements to track progress and gamification. It also stores decks and flashcards, including translations and learning history, to help users build their vocabulary. Temporary data, like translation results, is saved in LocalStorage during a session and is cleared when a new translation is made.

Data is collected through user interactions, such as filling out forms, saving flashcards, or answering quizzes. API calls to OpenAI process translations, while form submissions handle account setup. When a user saves a sentence or word, it is stored in MongoDB as a flashcard with its translation and explanation. Quiz answers, along with response times, are also saved to adjust learning schedules, though this plays a minor role. The app stores data as it is received without major modifications, following the set schema. The developer does not access user data directly and uses mock data for testing instead.

There have been some challenges in data collection. For example, the free tier of MongoDB database occasionally causes slow loading times, affecting performance. Additionally, improper API configurations could lead to accidental data loss. To prevent this, a backup database is planned for future improvements.

4.2 Ethical Considerations in User Data Management

Protecting user privacy is a key concern for the application. In earlier versions, privacy measures were not clearly defined, but now a checkbox has been added during registration, linking to a privacy page. This page explains what data is collected – such as account information, decks, and flashcards – and how it is used to improve the learning features. Users have control over some of their data, as they can delete their accounts or decks if they choose.

There are potential risks, although they are not fully identified at this point. Personal data breaches are not a major concern, as only usernames are checked for uniqueness, and no sensitive personal information is stored. Sentences users' input are sent to OpenAI's API rather than being processed by the application itself, so sensitive content is not directly handled by the application. Occasionally, OpenAI API may return errors, but these are typically due to glitches and are not usually tied to sensitive data. Currently, data is used only within the app for tasks like adjusting the learning algorithm based on analytics, and it is not shared with third parties.

The developer's experience learning Finnish didn't directly influence ethical considerations, and no specific regulations (such as school policies or data protection laws) were followed during the development process. However, basic ethical principles, such as keeping user data private and using it solely for the app's intended purpose, guided the development. Going forward, it may be beneficial to implement clearer privacy policies, including compliance with data protection laws (e.g., GDPR for users in Europe) to strengthen privacy protections.

4.3 Data Storage, Security, and Disposal

User data is stored in two places. Persistent data, such as account details, decks, and flashcards, is saved in MongoDB, which is hosted on Fly.io. Temporary data, like translated sentences and flashcards during a session, is stored in LocalStorage on the user's device and resets each time a new translation is made. To ensure security, passwords are encrypted using JSON Web Tokens (JWT), and cookies are used to maintain user logins. API keys are managed securely through environment variables, as outlined in the CI/CD setup, to prevent leaks.

Users have the ability to delete their data, either by removing specific decks or deleting their entire account, which will remove the data from MongoDB. Currently, there is no reset option, so once deleted, the data is not recoverable. If a user stops using the app, their data remains in MongoDB until it is manually deleted. In the future, an automatic deletion feature for inactive accounts (e.g., after one year of inactivity) could be implemented to clean up unused data.

So far, there have been no major security issues, but potential risks remain. CI/CD health checks and secure variable management via GitHub Actions help identify and address issues early, such as failed tests or setup problems. MongoDB's slow performance is a known challenge, and problems with data loss due to faulty API endpoints have highlighted the need for backups. To further improve security as the app grows, additional measures like stronger encryption or user data export options could be considered.

5 Empirical Part: Development and Implementation

5.1 Planning and Research

The idea for the language learning application came from a personal challenge: learning Finnish efficiently. Existing tools like Google Translate provided basic translations but lacked explanations for grammatical changes, making it difficult to understand sentence structures. While ChatGPT could break down sentences and explain word meanings, manually converting this information into study materials, such as flashcards, was time-consuming. This led to the idea of an automated tool that could generate structured learning content from translations, helping users grasp vocabulary and grammar more efficiently.

5.1.1 Identifying Key Learning Challanges

The main issue with conventional translation tools was their inability to clarify grammatical structures. Finnish, with its complex morphology and numerous word forms, required more than direct word-for-word translations. Learners needed explanations for why words changed based on tense, case, or sentence structure. Additionally, existing flashcard applications lacked integration with AI-powered explanations, requiring users to manually input and organize learning materials.

To address these gaps, research focused on:

- AI-assisted translation and grammar explanations – Exploring how OpenAI's API could provide structured breakdowns of sentences.
- Spaced repetition techniques – Studying how memory retention improves when vocabulary is reviewed at optimal intervals.
- Gamification in language learning – Investigating how rewards, progress tracking, and interactive exercises could enhance motivation.
- User experience in educational apps – Analyzing best practices for intuitive interfaces that reduce friction in daily use.

5.1.2 Initial Planning and Concept Development

The first phase of planning involved defining the app's core functionalities. Early ideas were recorded in a notebook, outlining:

- A translation input field for users to enter sentences.
- AI-generated explanations of words and grammatical structures.
- Automatic flashcard creation with word meanings, pronunciations, and example sentences.
- A simple interface for users to review and practice vocabulary.

To visualize these ideas, rough sketches were drawn for key app screens, including the translation page, flashcard decks, and quiz sections. These mockups provided a clear direction before development began.

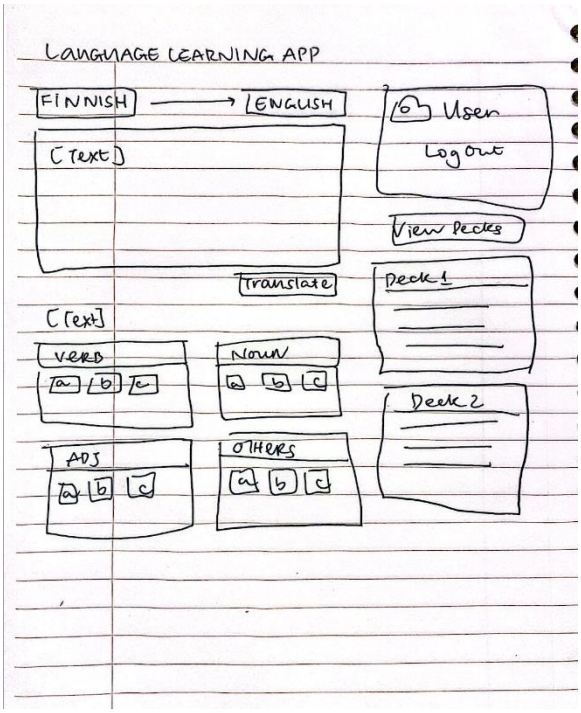


Figure 8: Initial hand-drawn sketch planning the main user interface, including translation input/output areas, user info, deck display, and word categorization sections.

5.1.3 Researching the Tech Stack

To ensure efficient development, the tech stack was chosen based on prior experience and existing knowledge:

- React.JS for building the user interface.
- Node.JS with Express.JS for backend operations and API handling.
- MongoDB for storing user-generated flashcards, user details, and learning progress.
- OpenAI API for generating translations and grammar explanations.
- TailwindCSS for styling, influenced by design patterns from similar learning apps.

This selection allowed the project to move forward without requiring extensive time to learn new technologies, keeping the focus on building core features.

5.1.4 Evolving the Concept

While the initial idea was simple – translating sentences and creating flashcards – the scope expanded as development progressed. Early testing showed the need for additional features, such as:

- Customizable flashcard decks for organizing vocabulary.
- Quizzes and interactive exercises to reinforce learning.
- User progress tracking to measure retention over time.

These refinements were shaped by both the developer's learning needs and insights gained from research into effective language learning methodologies

5.1.5 Summary

The planning and research phase laid the foundation for the app by identifying key language learning challenges and exploring AI-driven solutions. Early sketches, research into spaced repetition and gamification, and a well-defined tech stack helped streamline development. This phase ensured that the application was not just a generic translation tool but a structured learning platform designed to enhance vocabulary acquisition through AI-powered explanations and active recall techniques.

5.2 Development Process

Following the planning and research in Section 5.1, the development phase transformed initial sketches and concepts into a functional application. This section details the step-by-step implementation of key features, including AI-powered flashcards, spaced repetition, gamification, and personalization. The application was developed iteratively using the MERN stack (MongoDB, Express.JS, React.JS, and Node.JS with TypeScript), starting with translation functionality and expanding based on emerging requirements. The process involved utilizing mock data from the planning phase, hands-on coding, and continuous refinements. Code examples and screenshots are included to illustrate the development journey.

5.2.1 AI-Powered Flashcards

The AI-powered flashcards were the first major feature developed, starting with structuring the app's frontend (React.JS), and backend (Node.js, Express.js), and database (MongoDB). Development began with a simple UI layout based on an initial sketch – a translation box, user details, and a display area for generated flashcards. At first, placeholder text like "Translation here" was used to visualize the design before transitioning to mock data for proper layout testing.

The flashcard creation process followed a clear pipeline: user input → API call → data storage. A React form, styled with TailwindCSS and ShadcnUI, allowed users to enter a sentence. This input was sent to a Node.JS backend, which called OpenAI's API (via the OpenAI library) to generate flashcard content. OpenAI was selected in April 2024 for its affordability, well-documented API, and powerful GPT-4.0 model, following a cost-performance analysis by the developer. To ensure consistent output, the prompt was carefully crafted with a schema enforcing structured JSON responses.

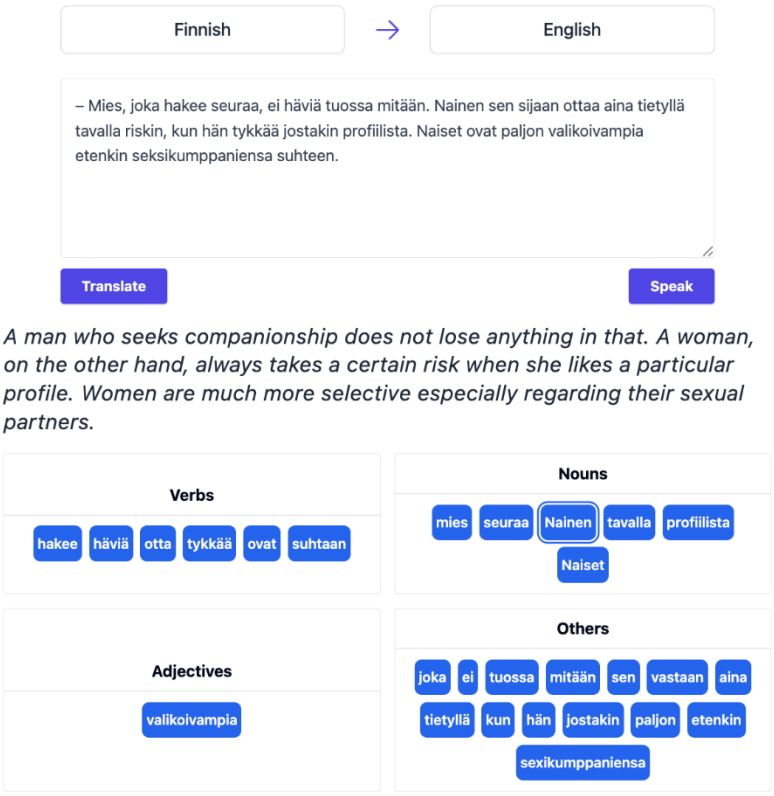


Figure 9: A view of the application's Index Page, highlighting the flashcards generated from the OpenAI API and categorized by their grammatical function.

The API returned data in a structured JSON format, including translation, pronunciation, word type, and lemmatization details, such as:

```
{
  "fi": "pitkän",
  "en": "long",
  "type": "adjective",
  "original_word": "pitkä",
  "pronunciation": "pit-kahn",
  "comment": "Adjective indicating a considerable length"
}
```

Figure 10: Example of the JSON data structure used to represent flashcard information.

This data was parsed in React to display interactive flashcards. Users could save them to MongoDB via an Express endpoint, while unsaved flashcards remained visible until a new sentence was entered. To optimize testing, early API responses were reused as mock data, reducing unnecessary API calls.

```
export const WordDetails: React.FC<{words: Word[]}> = ({ words }) => {
    const categories = [
        { title: 'Verbs', words: words.filter(word => word.type === 'verb' )},
        { title: 'Nouns', words: words.filter(word => word.type === 'noun' )},
        { title: 'Adjectives', words: words.filter(word => word.type === 'adjective') },
        { title: 'Others', words: words.filter(word => !["verb", "noun", "adjective"].includes(word.type))}
    ]
    return (
        <div className="mt-5 w-full max-w-4xl px-4 mx-auto">
            <div className="grid grid-cols-1 sm:grid-cols-2 gap-4">
                {categories.map((category, index) => (
                    <WordCategory key={index} title={category.title} words={category.words} />
                ))}
            </div>
        </div>
    )
}
```

Figure 11: The *WordDetails* component utilizes the parsed JSON data to create categorized flashcard sections (verbs, nouns, adjectives, others), as shown in the application's user interface.

Several challenges emerged during development. OpenAI's responses were initially inconsistent, occasionally missing fields or returning unexpected formats. This was resolved by refining the prompt with a strict schema (see Section 3.3.3). Database errors, such as failed saves, were fixed through connection debugging and error handling improvements.

To manage development efficiently, tasks were broken into small, prioritized user stories – e.g., "Add input box", "Call API", "Save card" - allowing for incremental progress.

Testing involved direct interaction with the app, entering sentences and reviewing flashcards. The UI initially struggled with responsiveness due to fixed pixel sizes, which was resolved by using TailwindCSS for adaptive design. Additionally, the first API-generated flashcards lacked detailed grammatical explanations. The prompt was adjusted to include richer explanations, enhancing the flashcards effectiveness for Finnish grammar learning. These refinements aligned with the goal of context-aware learning discussed in Section 5.1.

5.2.2 Spaced Repetition System

The spaced repetition system in this language learning application ensures users retain vocabulary effectively. It is integrated into the quiz system, categorizing flashcards into three stages:

- Not Studied – New words introduced for the first time
- Learning – Words under active review
- Completed – Well-learned words appearing less frequently

The system began with developing a multiple-choice quiz in React. Each questions presents a term from a deck with four answer choices – one correct definition and three random alternatives from the same deck. The quiz is built on the Learning Page, where a function fetches deck data from MongoDB via a Node.js endpoint and shuffles the options in the frontend.

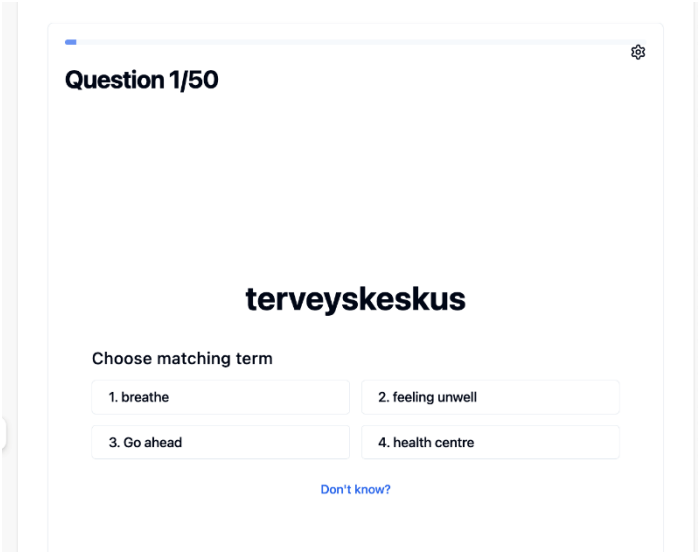


Figure 12: Example of a multiple-choice quiz question presented on the Learning Page, showing a term ("terveyskeskus") and four potential definitions.

After completing a quiz, the system records:

- Correctness – Right answers increase the interval before the next review
- Response Time – Slow responses indicate uncertainty and trigger earlier reviews.
- Review Count – Number of times a word was reviewed

From this, users receive a detailed recap displaying their correct answers, response times, and a ‘Next Review Date’ for words needing further practice, all of which are stored in the database.

```
{
  "cardsStudied": 4,
  "correctAnswers": 2,
  "quizType": "learn",
  "nextQuizDate": "2024-11-05T09:04:10Z",
  "reviewNumber": 0,
  "randomName": "blush_kite",
  "nextInterval": 1,
  "quizDetails": [
    {
      "q": "Term4",
      "yourAns": "Definition1",
      "correctAns": "Definition4",
      "correct": false,
      "score": 0,
      "time": 1944
    },
    {
      "q": "Term2",
      "yourAns": "Definition2",
      "correctAns": "Definition2",
      "correct": true,
      "score": 1,
      "time": 1393
    }
  ],
  "date": "2024-11-04T09:04:10Z"
}
```


Figure 13: JSON structure representing quiz results data stored in the backend, including cardsStudied, correctAnswers, quizType, nextQuizDate, and details for each question answered (q, yourAns, correctAns, correct, score, time).

Each word's progress is tracked with a hidden score in MongoDB, determining its study status: Not Studied (0–1), Learning (2–4), or Completed (5). A word reaches 'Completed' after five correct answers and is excluded from future quizzes, following Ebbinghaus' forgetting curve, which illustrates how memory declines without reinforcement. The backend handles scoring: correct answers increase a word's score by 1 (up to 5), while incorrect answers reset it to 0. After each response, the system updates the card's record with response time and accuracy data.

```
// Update score logic per card
if (correct) {
  card.cardScore = Math.min(card.cardScore + 1, 5);
} else {
  card.cardScore = 0;
}

// Determine learning state
if (card.cardScore === 0 && !card.learning) {
  card.learning = true;
}
if (card.cardScore >= 5) {
  card.learning = false; // Fully learned
}
```

Figure 14: Backend code snippet demonstrating the logic for updating a flashcard's *cardScore* based on correctness (incrementing up to 5 for correct, resetting to 0 for incorrect) and determining its learning state.

The review date logic was coded separately. New words default to a 1-day review, based on Ebbinghaus' 24-hour retention window. For reviews, the interval adjusts: 80 %+ correct doubles it (e.g., 2 → 4 days), while below 60 % halves it (min. 1 day), capped at 30 days for mastered words. If a review session contains only 5-scored cards, meaning all words are 'Completed', the session itself is marked as complete. This system draws from SuperMemo and Anki, with thresholds based on recall studies (Pavlik & Anderson, 2008; Capeda et al., 2006). The function calculates performance (correct answers ÷ cards studied) and updates MongoDB.

This review mechanism follows Spaced Repetition System principles, optimizing long-term vocabulary retention by adjusting review intervals based on individual performance. The algorithm's categorization aligns with the forgetting curve theory (Ebbinghaus, 1885), ensuring that words requiring more practice are reviewed more frequently, while well-learned material appears less often. This targeted approach minimizes forgetting and maximizes retention, key principles behind systems like SuperMemo and Anki.

The default interval is set to 1 day based on Ebbinghaus' Forgetting Curve, which suggests that newly learned information is rapidly forgotten unless reviewed within the first 24 hours. This phenomenon was first documented by Hermann Ebbinghaus (1885), who found that memory retention drops significantly within a day unless reinforced (Ebbinghaus, H. 1913, 54).

According to the forgetting curve, the optimal initial review period is between 10 to 24 hours after first exposure to reinforce memory. Setting the default review time to 1 day ensures that new words are revised before significant forgetting occurs.

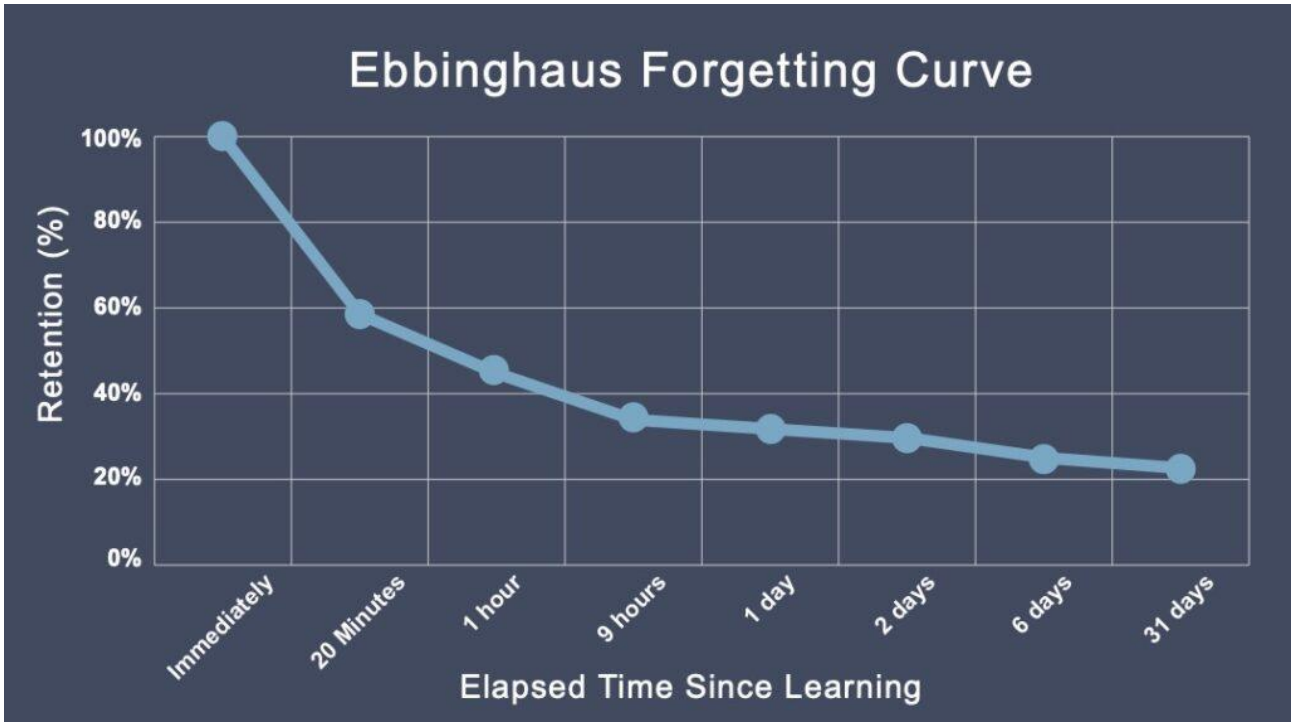


Figure 15: Visual representation of the Ebbinghaus Forgetting Curve, showing the exponential decline in memory retention over time without reinforcement.

To fine-tune how review intervals evolve over time, the algorithm applies performance-based thresholds. These thresholds (0.8 and 0.6) are drawn from cognitive science research and guide how intervals adjust based on recall success rates. The choice of these numbers aligns with

studies on optimal learning schedules in spaced repetition systems, such as those implemented in SuperMemo and Anki.

- 80 % Accuracy (0.8) - Double the Interval
  Research by Pavlik & Anderson (2008) found that successful rate recall at 80 % - 90 % accuracy is ideal for adjusting review schedules. High performance suggests strong retention, allowing the system to increase intervals without losing efficiency.
- 60 % Accuracy (0.6) - Reduce the Interval
  Studies in adaptive learning (Cepeda et al. 2006) found that recall rates below 60 % indicate weak retention. In such cases, decreasing the review interval helps prevent complete forgetting and reinforces difficult material sooner.
- The review interval is capped at 30 days, as in SuperMemo's SM-2 algorithm, to avoid excessive forgetting.
  This follows SuperMemo's SM-2 algorithm, where the maximum interval stabilizes around 30 days (about 4 and a half weeks) for well-learned material. Long intervals beyond this may lead to excessive forgetting.

```
function determineNextQuizDate(quizType, cardsStudied, correctAnswers) {
    const performance = correctAnswers / cardsStudied;
    let daysUntilNextQuiz = 1; // default for learn

    if (quizType === 'review') {
        if (performance >= 0.8) {
            daysUntilNextQuiz = Math.round(daysUntilNextQuiz * 2);
        } else if (performance < 0.6) {
            daysUntilNextQuiz = Math.max(1, Math.round(daysUntilNextQuiz * 0.5));
        }
        daysUntilNextQuiz = Math.min(30, daysUntilNextQuiz);
    }

    const nextQuizDate = new Date();
    nextQuizDate.setDate(nextQuizDate.getDate() + daysUntilNextQuiz);
    return nextQuizDate;
}
```

Figure 16: Backend function *determineNextQuizDate* calculating the interval (in days) until the next review based on quizType and performance (accuracy percentage), doubling the interval for >=80% accuracy and halving for <60% (min 1 day, max 30 days).

While the current algorithm uses fixed thresholds (e.g., 80 % to double the interval, 60 % to halve it) based on recall performance, it could be improved by:

- Dynamic Personalization – Adjusting intervals based on individual user performance rather than fixed values.
- AI-Based Predictions – Using machine learning models to optimize review intervals based on user learning patterns.
- Testing with More Users – Conducting controlled studies to fine-tune the accuracy thresholds for optimal retention.

These enhancements would allow the system to better adapt to diverse learning styles and maximize long-term retention.

Currently, the review algorithm reinforces learning by prioritizing words based on performance data. Words that are frequently forgotten are shown more often, while well-learned words are

spaced out. This adaptive scheduling follows the principles of spaced repetition, which recommend reviewing information at increasing intervals based on familiarity (Ebbinghaus, H. 1885, 54–63).

To effectively personalize each review session, the system evaluates user performance and classifies vocabulary into four main groups:

- Incorrectly answered words – These words require immediate reinforcement as errors indicate weak retention.
- Words answered correctly but slowly – Words that took longer than the user's average response time are considered weakly learned and prioritized.
- New or weakly learned words – Words with low scores (≤ 2) are scheduled for frequent review to prevent forgetting.
- Partially learned words – Words with mid-range scores (3–4) are included in review sessions but with lower priority

This categorization ensures that learners focus on weaker areas while preventing unnecessary repetition of well-known words.

```
const NextQuizCard: React.FC<NextQuizCardProps> = ({ quizData, averageTime, id }) => {
  const navigate = useNavigate();
  const details = quizData.quizDetails;

  const filterQuiz = (filterFn: (q: QuizDetail) => boolean) =>
    details.filter(filterFn).map(({ cardId, cardScore, question, correctAnswer }) => ({ cardId, cardScore, question,
correctAnswer }));

  const createQuizForReview = () => {
    const reviewArray = [
      ...filterQuiz(q => !q.correct), // Incorrect
      ...filterQuiz(q => q.correct && q.timeTaken > averageTime), // Correct but slow
      ...filterQuiz(q => q.cardScore <= 2), // Not studied / Ongoing
      ...filterQuiz(q => q.cardScore > 2 && q.cardScore <= 4 && q.correct), // Review again
    ];

    const shuffledArray = shuffleArray(reviewArray);
    navigate(`/review-page/${id}`, { state: { shuffledArray } });
  };
```

Figure 17: Frontend React component logic (*NextQuizCard*) filtering quiz results (details) to create a prioritized list (*reviewArray*) for the next review session, prioritizing incorrect answers, slow correct answers, and low-scored cards.

The algorithm first filters and maps quiz data to identify words needing priority. It then generates a personalized review session, selecting words based on performance and response time. This ensures that:

- Incorrect answers are reviewed multiple times within the same session.
- Slow responses are prioritized, reinforcing words where recall was uncertain.
- Unfamiliar words are revisited soon after introduction to prevent early forgetting.
- Partially learned words receive gradual reinforcement without excessive repetition.

The prioritization logic is grounded in cognitive learning principles and follows evidence-based strategies for optimizing recall:

- Error-based reinforcement: Studies show that immediate feedback and re-exposure to incorrect answers improve retention (Roediger & Butler, 2011)

- Response time as an indicator of uncertainty: Research indicates that longer response times suggest weaker memory consolidation, justifying early reinforcement (Kang, 2016)
- Early reinforcement of unfamiliar words: Based on Ebbinghaus' Forgetting Curve, newly learned material should be reviewed within 24 hours to prevent rapid decay (Ebbinghaus, 1885, 54–63)
- Controlled repetition of partially learned material: SuperMemo's SM-2 algorithm (Wozniak, 1994) suggests that material should be gradually spaced apart as mastery improves, avoiding unnecessary repetition

By integrating these principles, the application maximizes learning efficiency while preventing users from feeling overwhelmed by excessive reviews.

The approach to designing this algorithm was designed in an iterative manner:

- Initial Planning: A structured document was created to define how quizzes should function, what data needed to be stored, and how spaced repetition would be applied.
- Algorithm Design: Based on the document, a scoring system was implemented to categorize words correctly into different difficulty levels.
- Incorporating Response Time: Additional logic was added to prioritize words that were answered correctly but took longer than average response time.
- Iterative Testing: Edge cases were tested to ensure fairness, such as avoiding over-repetition of certain words while reinforcing weaker reels.

Appendix 4 contains the specific rules for scoring, prioritizing cards based on performance and response time, and modifying review intervals established during this initial planning phase.

While the current algorithm is effective, potential enhancements include:

- Testing with more edge cases: To ensure robustness, additional test cases can be included to stimulate different learning behaviors.
- Incorporating a Leitner system: A tier-based progression for flashcards based on user accuracy.
- Adjusting intervals with AI predictions: Using machine learning to predict optimal review times based on user learning behaviors.
- Introducing visual progress tracking: A dashboard that provides users with insights into their vocabulary retention rates.

These improvements can further enhance user engagement and ensure long-term knowledge retention in the application.

5.2.3 Gamification

Gamification was introduced to make learning more enjoyable and keep users engaged over time. It's split into two key parts: interactive learning tools (like the Matching Game and Flipping Page) and reward systems (XP, Streaks, and Badges) designed to build habits and boost motivation. These features were added after the flashcard and quiz systems were stable, aiming to add variety and fun to the learning experience. Development began with the interactive components, followed by XP and streak tracking to encourage daily logins. As of April 2025, leaderboard and streak freezes are still in progress, but the core features are live.

The core gamification mechanics implemented so far are described below:

- Matching Game

Feature Description: The Matching Game is a memory-based game where users are presented with 12 cards laid face down. Players flip two cards at a time, attempting to match vocabulary terms with their definitions. The game was implemented using React and TailwindCSS, and the vocabulary data is dynamically fetched from MongoDB.

Design Rationale: This interactive game was designed to enhance vocabulary recall through a fun and engaging activity that encourages users to actively engage with terms. It also adds variety to the traditional learning methods, making vocabulary review more enjoyable.

Expected Benefits: Offers a low-pressure, engaging way for users to reinforce their vocabulary knowledge through repeat exposure in a game format, improving retention and recall.

Purpose: To increase user engagement by providing an enjoyable learning experience that encourages regular practice and reinforces vocabulary acquisition.

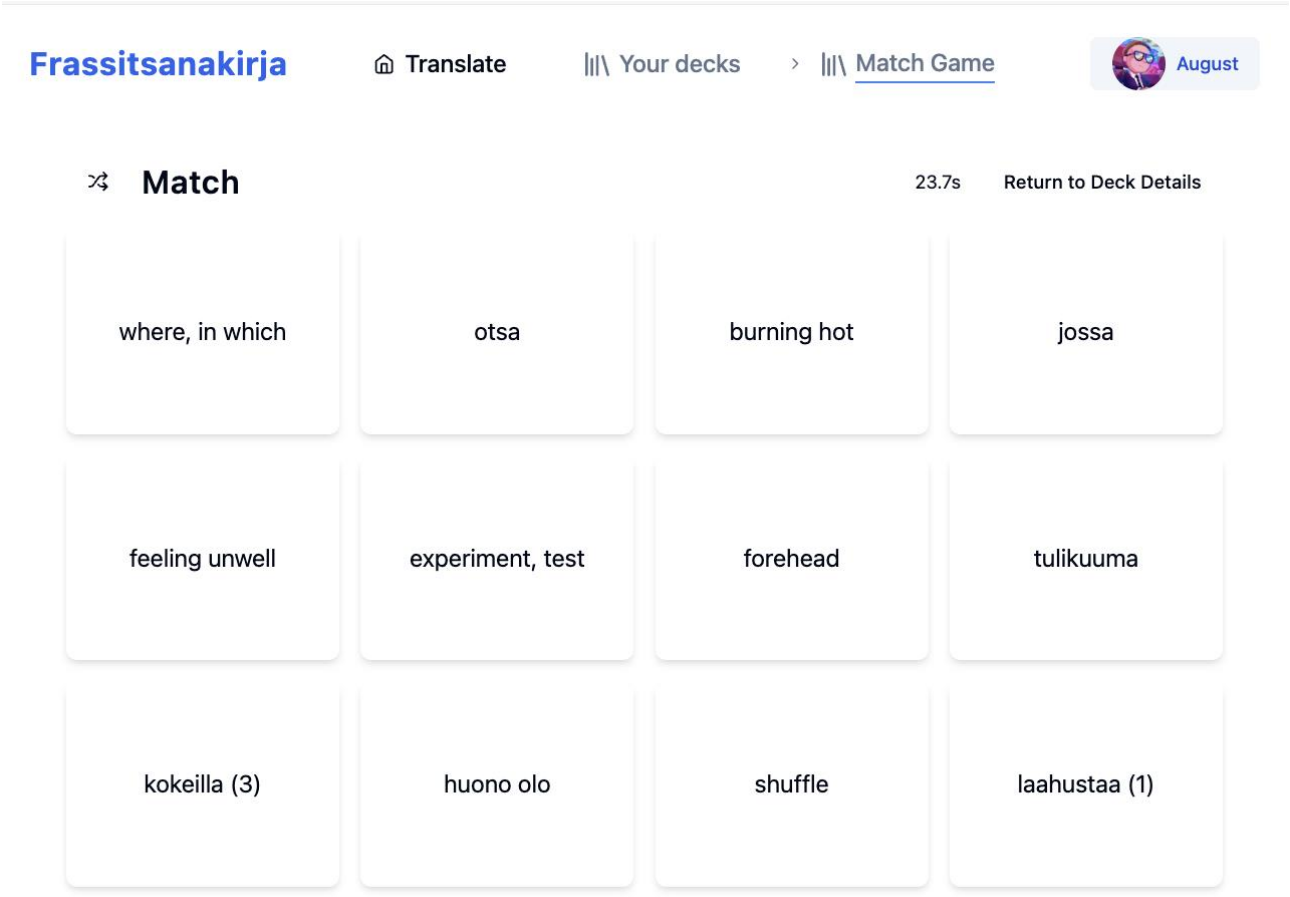


Figure 18: Screenshot of the interactive Matching Game interface, showing face-down cards, a timer, and navigation links.

- Flashcard Flipping Page

Feature Description: The Flashcard Flipping Page presents vocabulary terms one at a time in a digital card format. Built with React and TailwindCSS, and utilizing the “react-card-flip" library, this feature stimulates a physical flashcard interaction. Users can flip each card to reveal the definition, and optionally request a partial clue using the “Get a Hint” button. Navigation is user-driven, with

tick and cross buttons allowing progression through the shuffled deck. Flashcard data is dynamically loaded from MongoDB.

Design Rationale: This interaction model was introduced to reduce cognitive pressure commonly associated with quizzes. By mimicking traditional flashcard studying, the flipping mechanism supports self-paced exploration and reinforces memory through repetition. The hint functionality offers guided assistance without immediately disclosing the answer.

Expected Benefits: Flipping through cards in a low-stress setting encourages exploration and recall without performance anxiety. The optional hint supports retrieval practice, a learning technique shown to strengthen long-term memory by encouraging active recall (Roediger & Butler, 2011). Additionally, the physicality of the flip and interactive engagement has been linked to improved memory encoding due to embodied cognition – where physical interaction can enhance cognitive processing and retention (Wilson, 2002; Kontra et al., 2015)

Purpose: To provide an accessible, game-like alternative to formal assessments – encouraging more frequent, self-directed review sessions and supporting vocabulary retention via repetition and gentle prompting.

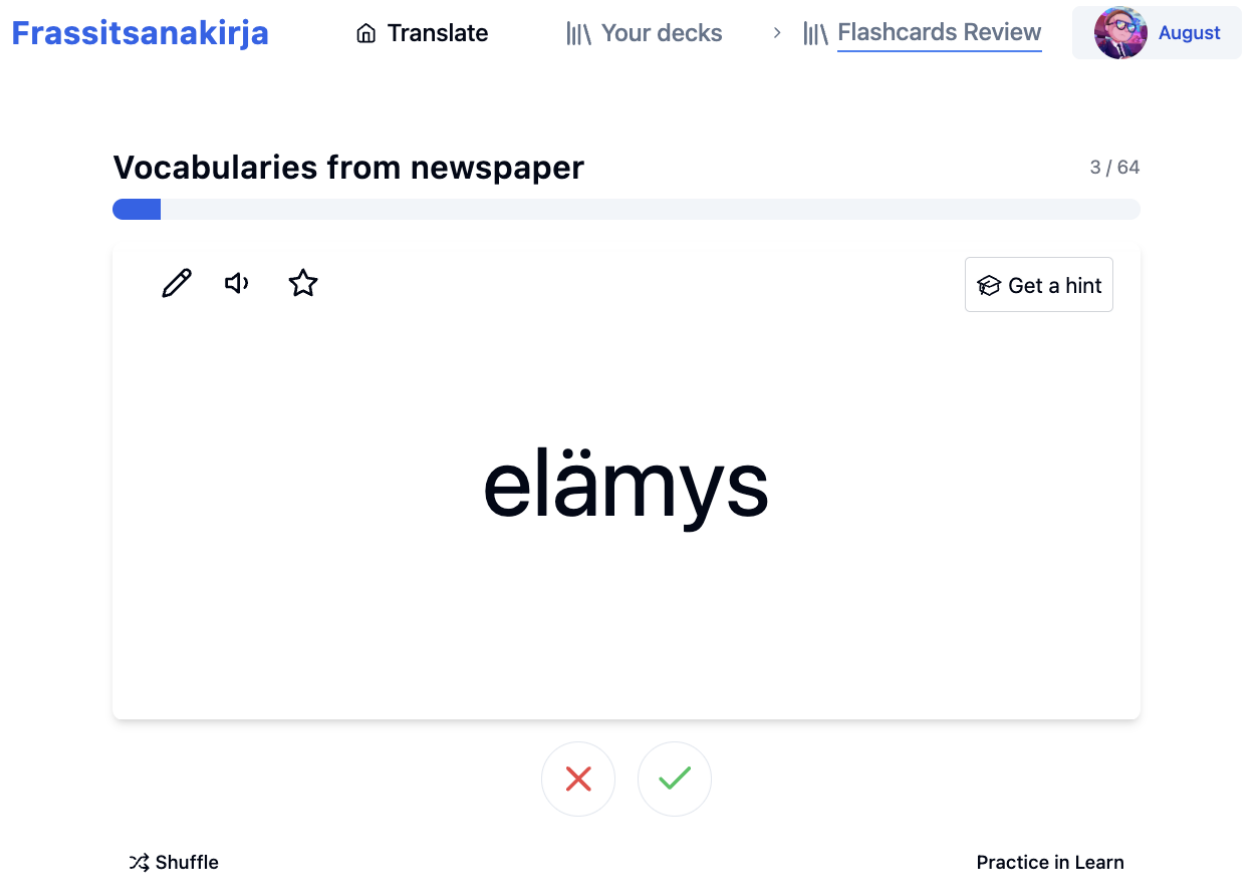


Figure 19: Screenshot of the Flashcard Flipping Page interface, displaying the front of a card ("elämys"), navigation buttons (tick/cross), shuffle option, and a "Get a hint" button.

- XP and Leveling System

Feature Description: The XP (experience points) system rewards users for engaging in learning activities such as completing quizzes, logging in daily, or translating sentences. Accumulated XP contributes to a user's overall level, which is visually represented in their profile. The XP calculation logic is implemented in the backend, and updated in real time through user actions.

Design Rationale: Inspired by gamification in popular language learning platforms, XP provides measurable progress that helps users feel a sense of advancement. Levels introduce short-term goals that sustain motivation across learning sessions, especially for users who enjoy progression tracking.

Expected Benefits: Research by Hamari et al. (2014) indicates that XP and leveling systems enhance engagement by providing clear progression and extrinsic rewards. In educational contexts, such systems encourage repeated interaction with content, supporting spaced repetition and long-term retention (Kang, 2016).

Purpose: To reinforce learning behaviors by linking effort with progression, thereby increasing retention and long-term usage of the app.

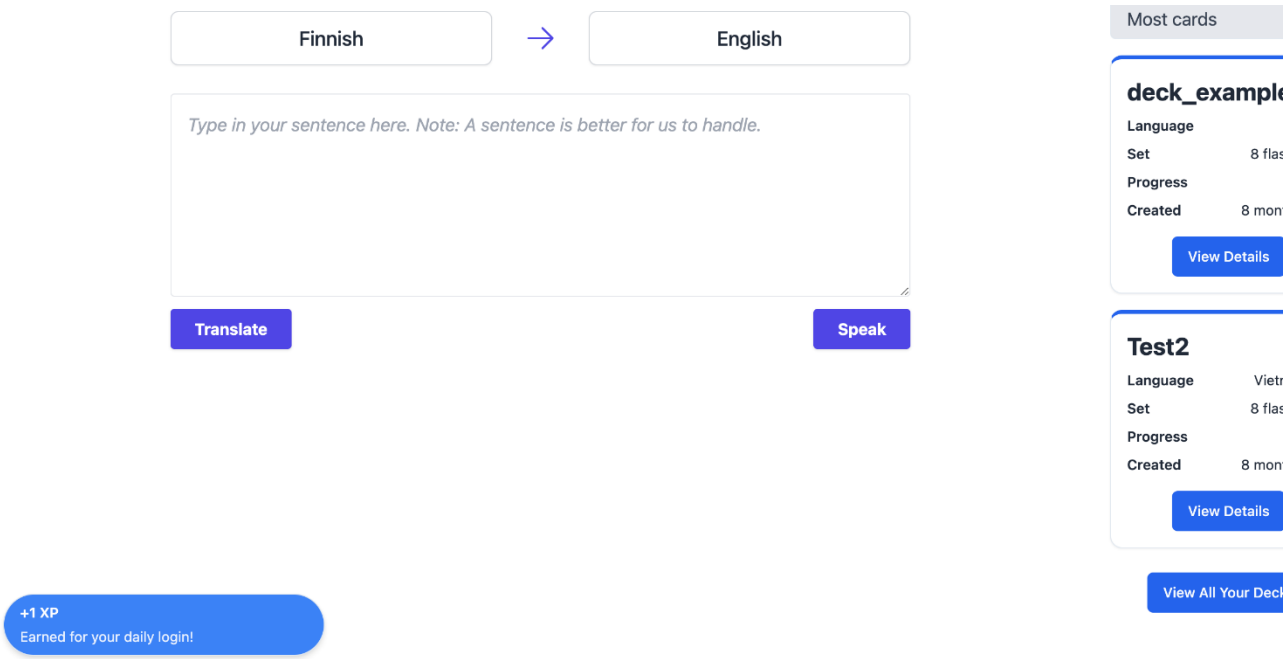


Figure 20: User profile section displaying the XP progress bar and current user level, illustrating the visual feedback for the leveling system.

- Daily Streaks

Feature Description: The Daily Streak system tracks consecutive days a user is active in the app. Each day the user completes a learning activity (e.g., quiz, flashcard session), their streak count

increases. If they miss a day, the streak resets. The current streak is prominently displayed on the dashboard.

Design Rationale: Streak-based systems are widely used to build habits by leveraging the psychological principle of loss aversion – users are more motivated to maintain their streak to avoid losing progress. (Blanco, 2024)

Expected Benefits: Daily streaks encourage regular engagement, aligning with the spaced repetition principle of reviewing material at optimal intervals (Ebbinghaus, 1885). Evidence 22 from Duolingo suggests that streaks improve user retention by establishing learning habits (Blanco, 2024), directly supporting the research question on how spaced repetition impacts vocabulary retention and engagement.

Purpose: To promote regular engagement and build long-term learning habits through positive reinforcement of consistent usage.

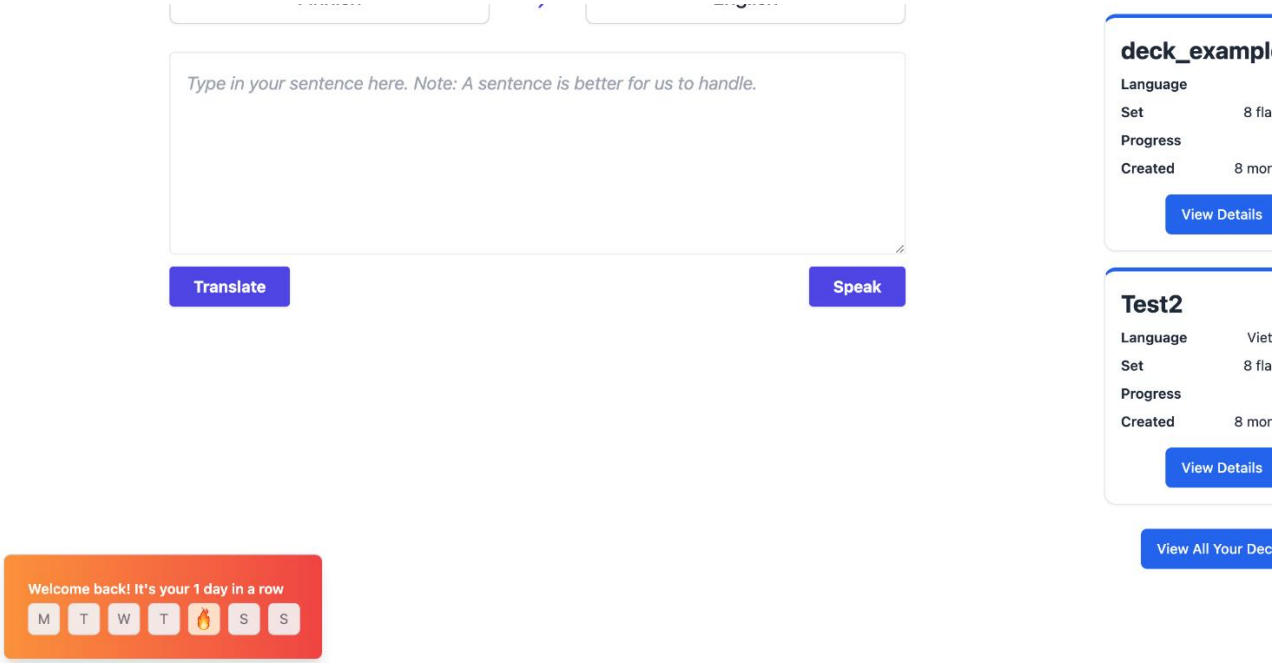


Figure 21: User profile section displaying the streak toast.

- Badges and Achievements

Feature Description: Users can earn badges for reaching key milestones – such as completing their first 10 quizzes, achieving a high streak, or mastering a set number of flashcards. These achievements are displayed in the user's profile as visual indicators of progress.

Design Rationale: This mechanic draws inspiration from competitive systems in fitness apps like Strava, where leaderboards encourage increased activity, and was adapted to foster a sense of community among language learners (Jojo, 2025). The regional/global split reflects the multilingual

focus of the application, allowing users to connect with peers locally or worldwide, enhancing the social aspect of learning

Expected Benefits: Competitive elements, such as leaderboards, have been shown to boost user activity and engagement in gamified systems (Hamari et al., 2014). By encouraging users to earn more XP through translations and quizzes, this mechanic indirectly increases interaction with AI-powered features and spaced repetition, addressing the research question on how such combinations impact engagement. Additionally, the social comparison may appeal to extrinsically motivated learners, broadening the application's appeal (Deci et al., 1999)

Purpose: Competitive elements foster a sense of community and encourage users to increase their activity, indirectly boosting engagement with the learning content

Figure 22: User profile section displaying earned badges and achievements, providing visual indicators of milestones reached.

The inclusion of competitive elements such as XP, streaks, and badges serves a dual purpose: not only do they provide external rewards, but they also create a sense of community and achievement, further motivating users to engage with the learning content. These features are rooted in educational psychology and cognitive science, aligning with well-established theories

The Matching Game and Flashcard Page features draw from the principle of retrieval practice, which emphasizes the importance of actively recalling information to strengthen long-term memory (Roediger & Butler, 2011). These low-pressure activities provide repeated exposure without the time constraints that traditional quizzes impose, reducing anxiety and increasing participation. Additionally, the act of flipping a card ties to embodied cognition, where physical interaction – like clicking and flipping – can enhance memory encoding (Wilson, 2002)

The XP and Leveling System leverages progress mechanics, which have been shown to foster sustained engagement by giving users visible signs of improvement (Hamari et al., 2014). Similarly, Daily Streaks build habit loops through loss aversion – users are more motivated to log in consistently to avoid losing progress (Blanco, 2024). These mechanics also align with spaced repetition (Ebbinghaus, 1885), encouraging daily activity that reinforces learning over time.

Lastly, Badges and Achievements provide extrinsic rewards that can spark initial motivation, especially when users hit milestones or complete challenges (Deci, Koestner, & Ryan, 1999). These reward systems are inspired by well-established learning apps like Duolingo, which combine immediate feedback with long-term progress incentives.

Those features are incorporated into the application with careful attention to user experience, ensuring that each element encourages consistent activity while also providing meaningful rewards. Each feature was developed with the goal of promoting sustained engagement through a balanced mix of challenge and reward.

Each feature was designed with a focus on user engagement, ensuring that each element would support continuous learning and interaction. The development of the matching game began with studying simple tutorials on classic memory games, which helped inform the logic for pairing cards. The UI was built using React and TailwindCSS, with cards dynamically loaded from the user's vocabulary deck stored in MongoDB via an Express endpoint. A shuffle function was implemented to randomize the cards, while click handlers compared the selections and provided immediate feedback.

```
// Inside useMatchGame.ts hook

const checkForMatch = () => {
    if (selectedCards.length === 2) {
        const [first, second] = selectedCards
        const firstCard = gameCards.find(card => card._id + '-' + card.type === first)
        const secondCard = gameCards.find(card => card._id + '-' + card.type === second)

        if (
            firstCard && secondCard && (
                (firstCard.engCard === secondCard.engCard) ||
                (firstCard.userLangCard === secondCard.engCard) // Assuming pairing logic based on either field
            )
        ) {
            // Match found
            setShowPenalty(false) // Reset penalty state if needed
            const newMatchedPairs = [...matchedPairs, first, second]
            setMatchedPairs(newMatchedPairs)
            if (newMatchedPairs.length === gameCards.length) {
                setIsGameCompleted(true) // Check if game is completed
            }
            setTimeout(() => setSelectedCards([]), 500) // Clear selection after a short delay
        } else {
            // No match
            setIncorrectPair([first, second]) // Highlight incorrect pair
            setTimeElapsed(prevTime => prevTime + TIME_PENALTY) // Apply time penalty
            setShowPenalty(true); // Show penalty indicator
            setTimeout(() => {
                setIncorrectPair([])
                setSelectedCards([])
                setShowPenalty(false) // Hide penalty indicator
            }, 800) // Clear selection and incorrect pair highlight after a delay
        }
    }
}
```

Figure 23: The *checkForMatch* function within the *useMatchGame* hook. This logic executes when two cards are selected, comparing their *engCard* or *userLangCard* properties to identify a pair. Upon a successful match, it updates the *matchedPairs* state, checks if the game is completed, and clears the selection. If no match is found, it briefly highlights the incorrect pair, applies a time penalty by updating *timeElapsed*, and resets the selection

The goal for the flashcard feature was to present one term at a time in an intuitive manner. Initial attempts using CSS transitions for flipping animations proved too glitchy, so the “react-card-flip" library was adopted for smoother animations. The feature allows users to toggle between hints and use ‘tick’ or ‘cross’ buttons to navigate through a shuffled set of cards. Like the Matching Game, this feature also fetches data from MongoDB via Express.

```
// Inside FlashcardPage.tsx component return statement

<motion.div
    key={currentCardIndex} // Ensures animation triggers on card change
    initial={{ x: "100%", opacity: 0 }}
    animate={{ x: 0, opacity: 1 }}
    exit={{ x: "-100%", opacity: 0 }}
    transition={{ type: "spring", stiffness: 200, damping: 30 }}
    className="w-full bg-white rounded-lg shadow-md overflow-hidden"
    onClick={handleFlip} // Allows flipping by clicking the card area
>
    <ReactCardFlip isFlipped={isFlipped} flipDirection="vertical">
        {/* Front of the Card */}
        <div key="front" onClick={handleCardClick}>
            <FrontCard
                word={cards[currentCardIndex]?.userLangCard}
                hint={hint}
                onGenerateHint={handleGenerateHint}
            />
        </div>

        {/* Back of the Card */}
        <div key="back" onClick={() => setIsFlipped(!isFlipped)}>
            <BackCard word={cards[currentCardIndex]?.engCard} />
        </div>
    </ReactCardFlip>
</motion.div>

// ... (Buttons for navigation would follow this)
```

Figure 24: Solution to the animation challenge described earlier, utilizing *ReactCardFlip* to provide a smooth flashcard flipping experience controlled by user clicks.

The XP system was designed as a core motivator, where users earn 1 XP for each translation and login, with additional XP for future quiz interactions. The leveling system uses the formula “*Math.floor(Math.sqrt(xp / 100)) + 1*”, which was intentionally chosen to control the rate of progression as users accumulate XP. This formula works by first taking the square root of the XP value, which reduces the impact of large XP values, thus slowing the rate of level advancement as the user progresses.

```
// Helper function to calculate level based on XP
function calculateLevel(xp) {
    return Math.floor(Math.sqrt(xp / 100)) + 1;
}
```

Figure 25: The *calculatelevel* helper function implementing the formula used to convert accumulated XP into user levels, ensuring a slowing progression curve as described in the text.

The “Math.floor()” function ensures that the result is rounded down to the nearest integer, and adding 1 ensures that the user always starts at level 1, even at the beginning of the XP journey. This approach ensures that early levels are achieved quickly, encouraging new users to engage and experience progress, but as XP grows, the leveling curve becomes slower, which prevents users from reaching higher levels too quickly. It helps maintain long-term engagement by making each level up feel more rewarding as the user continues learning.

To implement this, the MongoDB schema was updated to include fields for both XP and level. A new */award-xp* endpoint was created to handle XP updates, calculating and storing the new level each time XP is earned. A progress bar in the React front end dynamically reflects the user’s current XP and level, providing real-time feedback as users progress through the system.

```
// Inside the POST /award-xp endpoint

// Add XP (adjusting for potential multiplier)
const adjustedXp = applyXPMultiplier(user, xpAmount);
user.xp += adjustedXp;
user.weeklyXP += adjustedXp; // Assuming weeklyXP is also tracked

// Calculate new level using the helper function
const newLevel = calculateLevel(user.xp);

// Update user's level if they leveled up
if (newLevel > user.level) {
    user.level = newLevel;
    // Optional: Add logic for level-up rewards here (e.g., streak freezes)
    if (newLevel <= 10) {
        user.streakFreezes += 1;
    }
    // ... other level-up rewards
}

// ... (rest of the endpoint logic, including saving the user)
```

Figure 26: Snippet from the */award-xp* backend endpoint illustrating how earned XP is added to the user’s total and the *calculateLevel* function is invoked to determine and store the potentially new user level in the database.

The streak feature encourages consistent daily use by tracking two fields in the user schema: “currentStreak” and “lastActiveDate”. On each login, the backend compares the stored “lastActiveDate” with the current date. If the difference is exactly one calendar day, the “currentStreak” is increased by one. If the difference is more than one day, the streak is reset to zero. To avoid issues with time zones and edge cases (e.g., logging in at 23.00 one day and 00.30 the next), the comparison uses fixed 24-hour calendar days rather than timestamps.

Streak rewards are given at specific milestones, such as 3, 7, and 30 days (about 4 and a half weeks). When these thresholds are reached, the backend includes a reward notification in the response. On the frontend, toast notifications are triggered using ShadcnUI to inform users about their current streak status and any unlocked rewards. Streak data is pulled from MongoDB and refreshed on each login.

```
// Inside the POST /award-xp endpoint, after streak calculation

// Streak rewards based on current streak milestones
if (user.currentStreak === 3) {
    const bonusXp = applyXPMultiplier(user, 50); // Calculate potential bonus XP
    user.xp += bonusXp;                          // Add bonus XP
    responseData.streakReward = { type: 'xp_boost', amount: bonusXp };
} else if (user.currentStreak === 7) {
    // Award 1.2x XP multiplier for 24 hours and a streak freeze
    user.xpMultiplier = 1.2;
    user.xpMultiplierExpiration = new Date(today.getTime() + 24 * 60 * 60 * 1000);
    user.streakFreezes += 1;
    responseData.streakReward = {
        type: 'xp_multiplier',
        multiplier: 1.2,
        expires: user.xpMultiplierExpiration,
        extra: { type: 'streak_freeze', amount: 1 } // Include extra reward info
    };
} else if (user.currentStreak === 30) {
    // Award a specific achievement for a 30-day streak
    // (Check added below to prevent duplicates if run multiple times on day 30)
    if (!user.achievements.some(a => a.name === '30-Day Streak')) {
         user.achievements.push({
            name: '30-Day Streak',
            description: 'Maintained a 30-day streak'
         });
         responseData.streakReward = { type: 'achievement', name: '30-Day Streak' };
    }
}

// ... rest of endpoint
```

Figure 27: A backend logic for awarding specific rewards upon reaching streak milestones (3, 7, and 30 days) within the */award-xp* endpoint. Rewards include bonus XP, temporary XP multipliers, streak freezes, or unique achievements.

Badges recognize key learning milestones to boost user motivation. Examples include earning the first translation, completing 10 quizzes, or maintaining a high streak. The MongoDB user schema includes an “achievements” array that stores unique badge IDs. On key user actions, the backend checks if a new badge condition is met and verifies whether it has already been awarded. If the badge is new, it is added to the “achievements” array and stored in the database.

```
// Inside the POST /award-xp endpoint

// Define achievements triggered by specific activities
const achievementMap = {
    'translate': { name: 'First Step', description: 'Translate your first phrase!' },
    'quiz_complete': { name: 'Quiz Champion', description: 'Complete a quiz' },
    'daily_login': { name: 'Daily Visitor', description: 'Log in for the first time today!' }
    // Add more activities and achievements here
};

// Check if the current activity triggers an achievement
if (achievementMap[activity]) {
    const achievement = achievementMap[activity];
    let shouldAward = false;

    if (achievement.name === 'Daily Visitor') {
        // Special check for daily login achievement
        const isFirstLoginToday = !user.lastActiveDate || new Date(user.lastActiveDate).toDateString() !== today.toDateString();
        // Award only if not already earned AND it's the first login activity of the day
         if (!user.achievements.some(a => a.name === achievement.name) && isFirstLoginToday) {
            shouldAward = true;
        }
    } else {
        // Standard check: Award if not already earned
        if (!user.achievements.some(a => a.name === achievement.name)) {
            shouldAward = true;
        }
    }

    // Add achievement to user's array if conditions met
    if (shouldAward) {
        user.achievements.push({
            name: achievement.name,
            description: achievement.description,
            // dateEarned is added by default by Mongoose schema
        });
         // Optionally add achievement info to responseData for frontend notification
        responseData.newAchievement = achievement;
    }
}

// ... (XP-based badge logic could also be shown here or separately)
```

Figure 28: Snippet from the */award-xp* endpoint demonstrating how activity-based achievements are awarded. It uses an *achievementMap* to define triggers and pushes new achievements to the user's *achievements* array after checking to prevent duplicates.

To prevent duplicates, the backend logic ensures each badge is only added once. On the front end, the profile page fetches the achievements list from the backend and displays it. This check occurs only when a badge-triggering event happens, reducing unnecessary fetches and improving performance. The badge system provides visual feedback and acts as a log of long-term progress.

```
// Inside UserSchema definition from models/User.js

const UserSchema = new Schema({
    // ... other fields like name, username, xp, level, streak ...

    // Achievements array stores objects with achievement details
    achievements: [{
        name: { type: String, required: true }, // Name acts as unique identifier
        dateEarned: { type: Date, default: Date.now }, // Automatically records when earned
        description: { type: String }
    }],

    // Badges array stores objects based on milestones (e.g., XP)
    badges: [{
        name: { type: String, required: true }, // Name acts as unique identifier
        tier: { type: String, enum: ['Bronze', 'Silver', 'Gold', 'Platinum'], required: true },
        dateEarned: { type: Date, default: Date.now }
    }],

    // ... other fields like weeklyXP, region ...
});
```

Figure 29: Mongoose schema definition for the *achievements* and *badges* arrays within the User model. This structure facilitates storing detailed milestone information (name, description, date, tier) and enables checks to prevent duplicate entries, as described in the text.

Throughout development, several technical challenges emerged that helped shape the final implementation. Flashcard flipping animations were initially problematic due to choppy CSS transitions. After experimenting with different solutions, the *react-card-flip* library was chosen, greatly improving performance. The matching game worked well from the start, though adapting the pairing logic from online tutorials required rewriting parts of the code to improve state management and handle asynchronous state updates correctly.

The XP system also presented issues: users were able to farm XP by repeatedly refreshing the page. This was resolved by introducing a *hasAwardedXpToday* flag stored in *localStorage*, ensuring XP is awarded only once per day. Streak tracking faced bugs when users logged in near midnight, causing unexpected resets or skipped increments. The logic was updated to compare calendar days using 24-hour intervals rather than exact timestamps. For badges, users were initially able to earn the same achievement multiple times. This was fixed by strengthening backend checks to ensure each badge is granted only once. A more significant issue surfaced when development and production environments shared the same MongoDB cluster. This caused test data to leak into the live application, highlighting the importance of clearly separating environments to protect user data and maintain system integrity.

```
// Inside useGamification.ts hook

// Initialize state by checking localStorage if XP was already awarded today
const [hasAwardedXpToday, setHasAwardedXpToday] = useState(() => {
    const lastAwardDate = localStorage.getItem('lastXpAwardDate');
    const today = new Date().toDateString();
    // If an award date is stored and matches today's date, return true
    return lastAwardDate === today;
});

// State to prevent concurrent requests
const [isAwarding, setIsAwarding] = useState(false);
// ... other state and toast manager hook ...

const awardDailyLoginXp = async () => {
    // Prevent awarding XP if already awarded today or if an award is in progress
    if (hasAwardedXpToday || isAwarding) return;

    setIsAwarding(true); // Set flag to indicate awarding process started
    const today = new Date();
    try {
        // Make API call to the backend /award-xp endpoint
        const response = await axios.post("/gamification/award-xp", {
            xpAmount: 1, // Award 1 base XP for daily login
            activity: 'daily_login',
        });
        const data = response.data;

        // ... (Handle showing toasts based on response) ...

        // On successful award, store today's date in localStorage
        localStorage.setItem('lastXpAwardDate', today.toDateString());
        // Update state to reflect that XP has been awarded for today
        setHasAwardedXpToday(true);
        await refreshUserStats(); // Refresh user stats after successful award

    } catch (error) {
        console.error("Error awarding XP: ", error);
        showErrorToast("Failed to award XP. Please try again.");
    } finally {
        setIsAwarding(false); // Reset awarding flag regardless of success/failure
    }
};

// ... rest of the hook
```

Figure 30: Frontend logic within the *useGamification* hook implementing the fix for the XP farming issue. It uses *localStorage* (*lastXpAwardDate)* and component state (*hasAwardedXpToday)* to ensure the *awardDailyLoginXP* function only executes its API call once per calendar day, preventing users from gaining daily login XP multiple times by refreshing.

These experiences directly informed plans for future improvements. Leaderboards are currently being developed to show XP-based global and regional rankings. A new *weeklyXP* field will allow weekly resets, and the UI will render sortable tables based on user location. Inspired by apps like Duolingo, a streak freeze feature will also be added, allowing users to earn “freeze tokens” after maintaining a 7-day streak. If a day is missed, a token will be used automatically to preserve progress. Quiz features will soon be integrated with the XP system, awarding 1–3 XP per correct answer based on difficulty or personal success rate. This will require expanding the */award-xp* endpoint to process quiz scores and context. In the long term, the developer plans to experiment with advanced reward mechanics such as team-based challenges, AI-personalized XP boosts, and A/B testing to evaluate the motivational impact of streaks, badges, and other elements in the learning experience.

5.2.4 Personalization

Personalization was introduced to enhance user experience, divided into AI-Tailored Flashcards and a Testing page. These features emerged after the integration of gamification, focusing on refining flashcards and adapting vocabulary tests. AI-Tailored Flashcards modify card meanings to ensure accuracy, while the Testing page provides custom quizzes that go beyond traditional multiple-choice formats. Both utilize AI to adapt to user needs, with the majority of the coding done in

the UI and backend updates for saving any changes. As of April 2025, these features are live, with plans for future expansion.

The AI-Tailored Flashcards feature checks the accuracy of flashcards by considering broader contexts. Initially, the Deck Details page was built with React and TailwindCSS, displaying a list of cards fetched from MongoDB via Express. The OpenAI API setup from previous versions was reused to test prompts, generating structured JSON data (term, definition, adjustments). All flashcards from a deck are sent to OpenAI's API for review, with the AI flagging and suggesting corrections where necessary. For example, the term "kirkko" might be changed from "church" to "cathedral" when referring to a specific type of church in a historical context, providing greater clarity on the meaning intended. To offer additional reference options, web scraping from dictionaries was integrated using Node.js, allowing users to view dictionary definitions alongside AI suggestions. A review UI displays these options, and users can select the changes they agree with. Once confirmed, the updated cards are sent back to the backend to modify the database, reflecting the new schema.

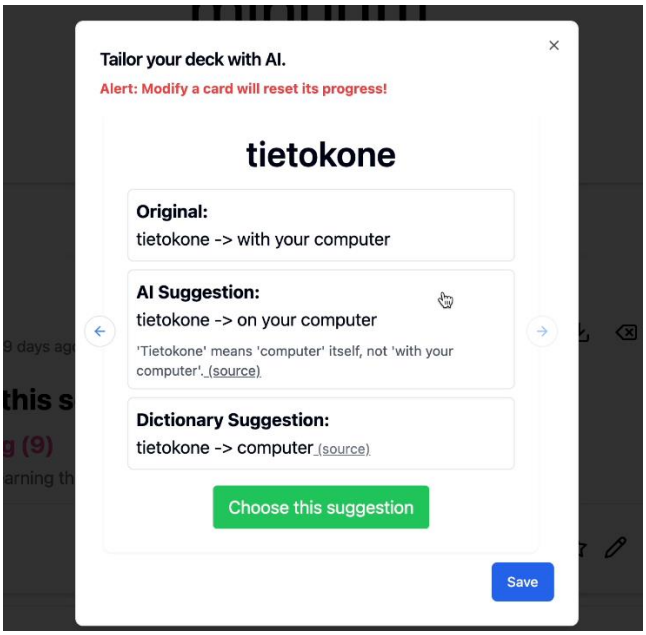


Figure 31: AI-Tailored Flashcards interface showing the original card ("vuoro -> turn, shift"), AI-suggested refinement with explanation, dictionary suggestion, and button to accept a suggestion.

The Testing Page was developed to create adaptive vocabulary tests, driven by a desire for greater variety in quizzes. A function was written to send deck vocabulary to OpenAI's API, prompting it to generate passages with fill-in-the-blank questions. The UI, built with React and TailwindCSS, presents these passages alongside scrambled word and synonym matching options. OpenAI selects 5 – 10 words from the user-selected vocabulary that complement each other to form a coherent story, ensuring context remains intact.

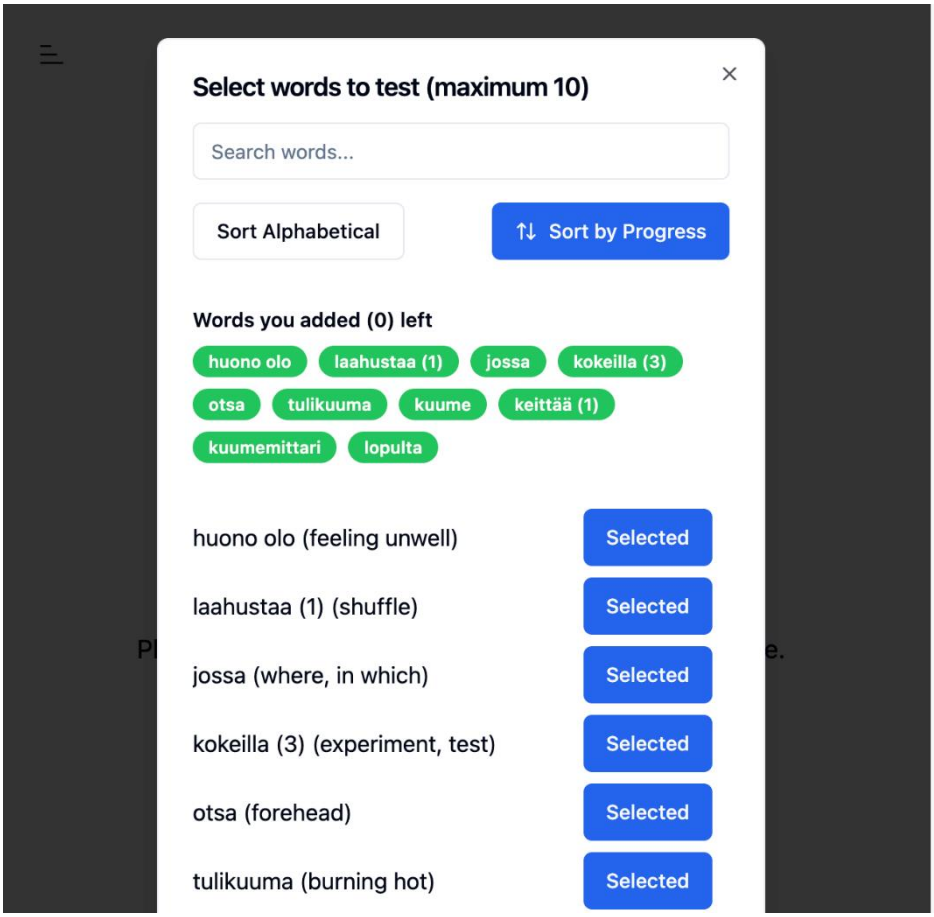


Figure 32: The interface for choosing vocabularies for the Testing Page, either by the progress of learning and users' favors.

Personalization adapts the experience to user performance, combining AI-driven content and analytics. Inspired by Vietnam's assessment model, which values varied question types for reinforcing learning (Nguyen, 2019), the test is structured into three sections:

- Passage: An AI-generated paragraph with fill-in-the-blank questions, based on selected vocabulary and tailored to match its CEFR level (e.g., A1 vocabulary yields A1-level text). Optional aids like translation, context, and vocabulary hints are hidden by default but accessible.

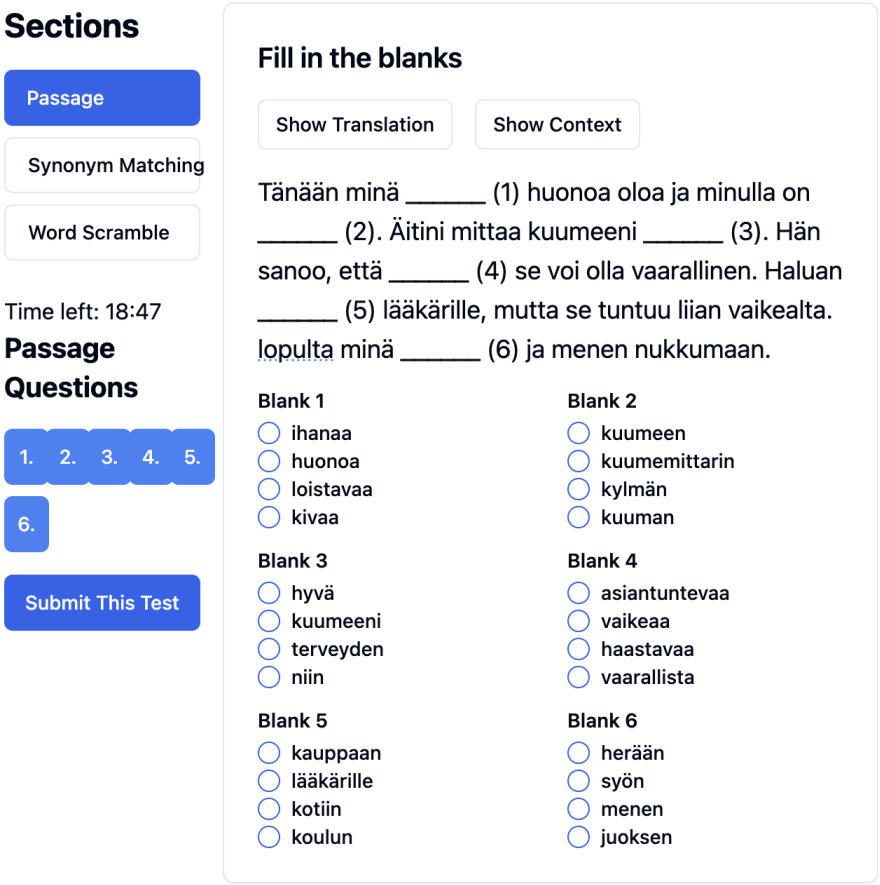


Figure 33: Testing Page - Passage section displaying an AI-generated paragraph with numbered blanks and multiple-choice options for Blank 1.

- Word Scrambling: Algorithm-generated challenges where users rearrange letters to form correct words.

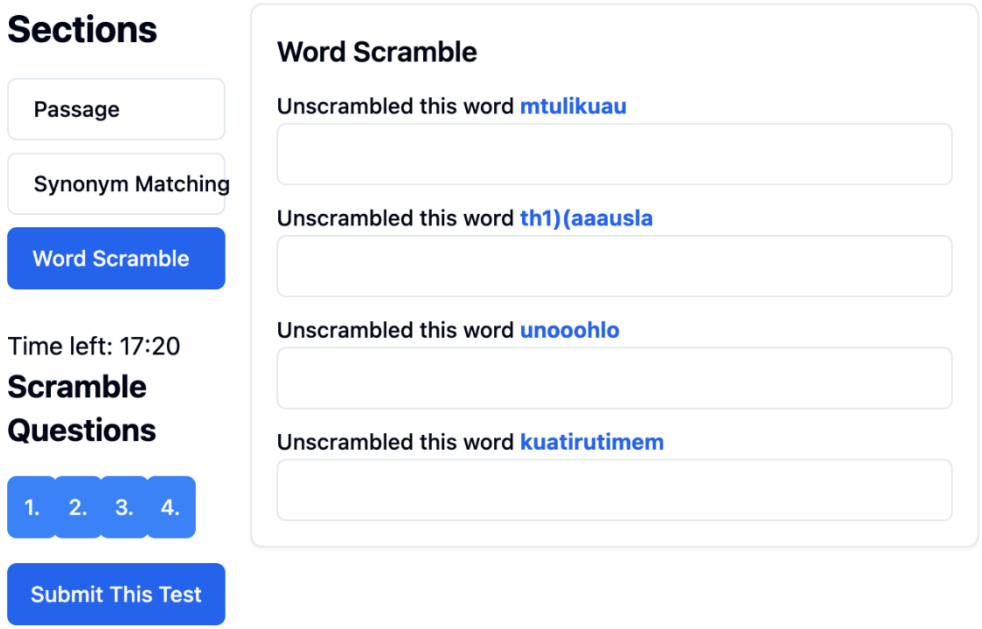


Figure 34: Testing Page - Word Scramble section showing letters to be rearranged to form a correct vocabulary word.

- Synonym Matching: Multiple-choice questions where users identify synonyms, testing word recognition and nuance understanding.

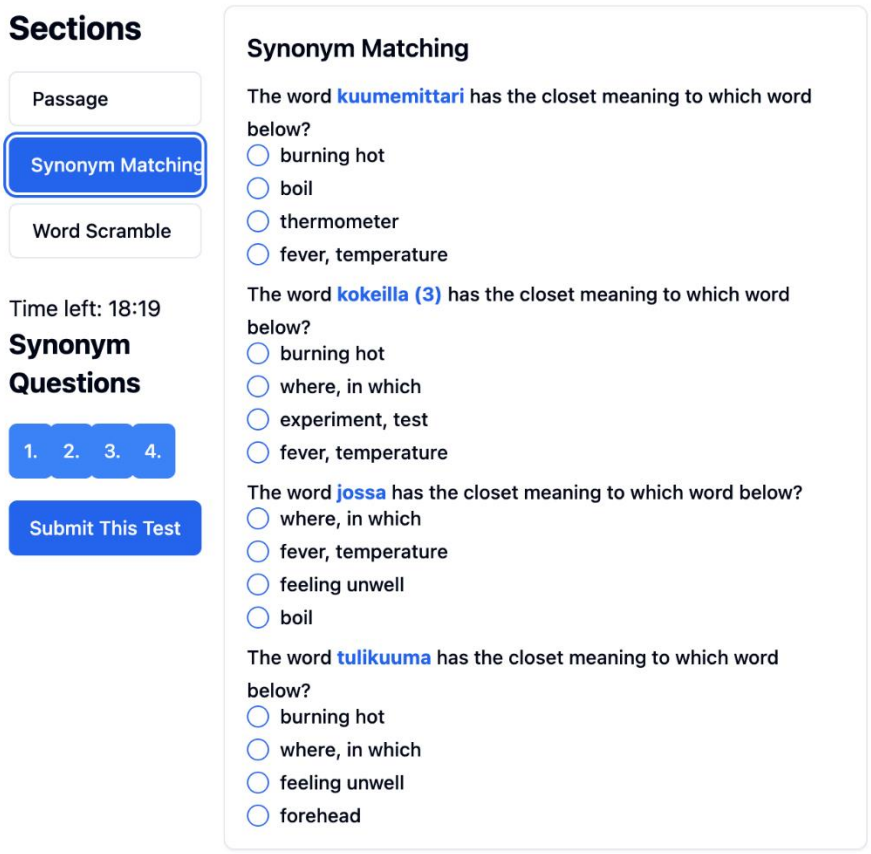


Figure 35: Testing Page - Synonym Matching section presenting a word ("ilo") and multiple-choice options for its synonym.

Users select 10 words to test – either manually or by auto-selected based on alphabetical order or performance metrics like correctness rate. A 20-minute timer, implemented with the *react-count-down* library, persists through rerenders to simulate time-limited testing conditions.

Post-test, a React component displays detailed statistics – correctness, time, and score – fetched from MongoDB via a Node.JS endpoint. Though difficulty adjustment is currently basic, it aligns with vocabulary level and lays groundwork for more refined personalization.

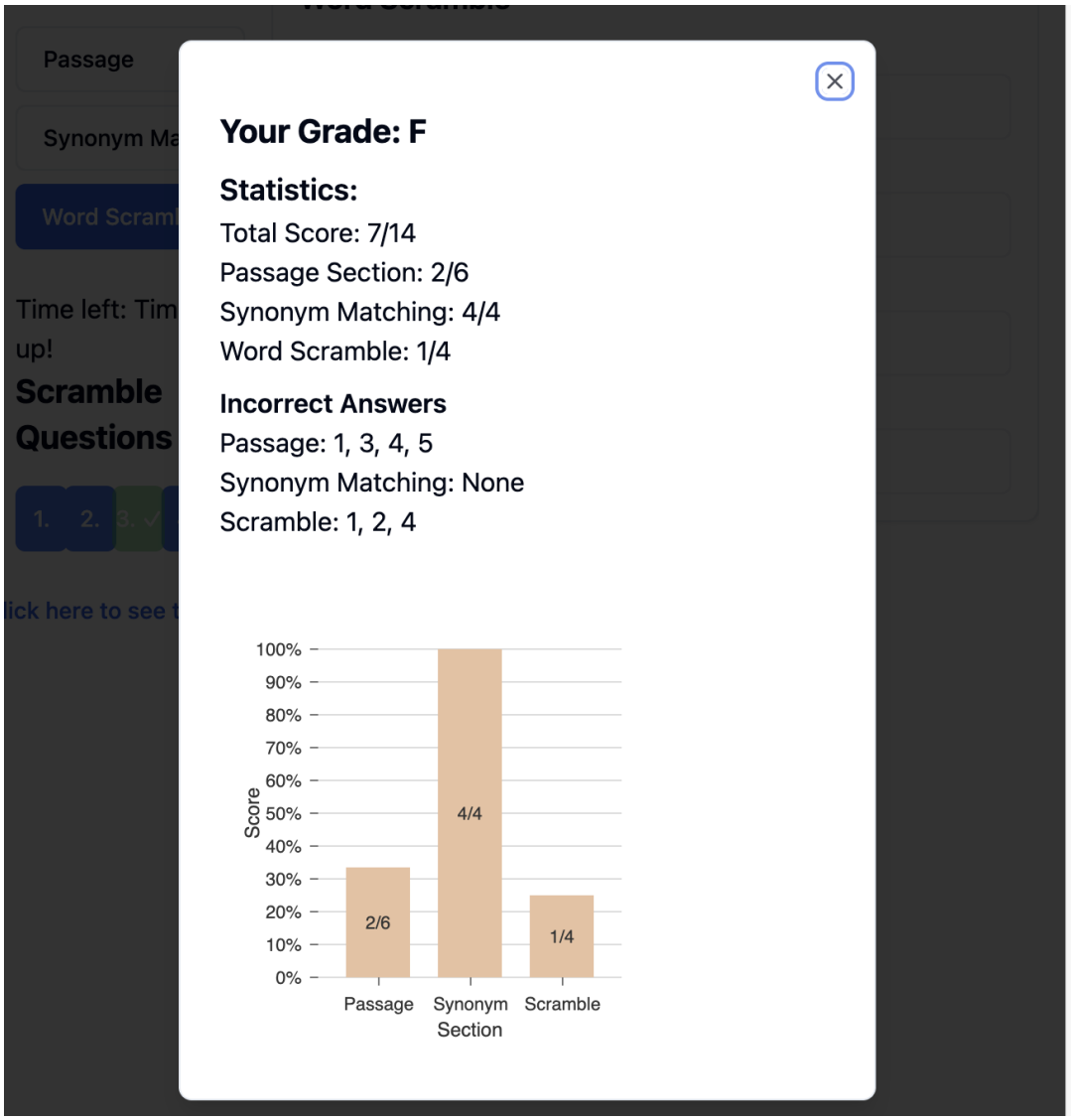


Figure 36: Post-test results display showing statistics like correctness percentage, time taken, and overall score.

Personalization was implemented to optimize language learning by adjusting difficulty and content to the user's needs. The flashcard refinement relies on generative learning theory, where restructuring and reviewing content improves memory and understanding (Fiorella & Mayer, 2020). The Testing Page adapts to user performance and follows principles from the zone of proximal development, keeping challenges appropriately difficult. Its structure – passage, scrambling, and matching – is inspired by Vietnam's diversified assessment model (Nguyen, 2019), aiming to support long-term retention with minimal user effort.

The development of personalization features followed a problem-first approach: how to improve content quality and create more engaging, adaptive tests. Rather than building from scratch, the existing OpenAI integration was extended with features that emphasized contextual relevance,

minimal UI friction, and a clear feedback loop for learners. Iterative prototyping was used to quickly validate prompt behavior and interaction patterns before full-scale implementation, especially in the review and testing flow.

To handle content transformation, custom hooks were introduced to keep the logic modular and composable. For instance, "*useModifiedCards*" was responsible for orchestrating the API call to OpenAI, then post-processing the results with dictionary data. This allowed the original user-submitted card to be paired with both AI-generated alternatives and an explanation string. The OpenAI API returned data in the following format:

```
{
  "_id": "663dc40e7d1e5facfbbaf407",
  "engCard": "service",
  "explanation_string": "In Finnish, 'messu' refers specifically to a
  Lutheran church service, while 'service' has a broader meaning encompassing
  any kind of service (e.g. customer service, public service).
  https://evl.fi/tervetuloa-kirkkoon/messu-eli-jumalanpalvelus/"
}
```

Figure 37: Example JSON structure returned by the OpenAI API for the AI-Tailored Flashcards feature, including the original term, suggested term, explanation, and source link.

This format helped reduce ambiguity in AI-suggested alternatives by including a human-readable explanation and an external reference link. The hook then mapped this response back to the user's original card:

```
const modifiedCardsPromises = modifiedCards.map(async (card) => {
  const originalCard = cards.find(c => c._id === card._id);
  const explanation = extractExplanation(card.explanation_string || '');
  const dictionarySuggestion = await getDictionarySuggestion(originalCard?.userLangCard |
  return {
    ...originalCard,
    aiEngCard: card.engCard,
    explanation,
    dictionarySuggestion,
    chosen: false,
  };
});
```

Figure 38: Frontend logic mapping the OpenAI API response (Figure 36) to the application's internal card structure, combining original and suggested data.

To simplify user interaction during review, a carousel-style interface was introduced, enabling easy toggling between the original submission, the AI alternative, and dictionary-based suggestions. The

logic for handling option selection was centralized in a separate hook, "*useHandleButtonClick",* which cleanly tracked the currently chosen card variant:

```
const handleSuggestionClick =
(selectedSuggestion: string,
setSelectedSuggestion: any,
suggestionType: string,
index: number) =>
{
  const suggestionId =
    suggestionType === 'original' ? 'original' : `${index}-${suggestionType}`;
  setSelectedSuggestion(suggestionId === selectedSuggestion ? '' : suggestionId);
};
```

Figure 39: Custom React hook *useHandleButtonClick* managing the state for the carousel-style review interface, tracking the user's selected card variant (original, AI, dictionary).

This UI pattern allowed users to compare choices in a low-friction environment and helped reduce cognitive load, especially in sessions with many cards to review. To further enhance the user experience during longer sessions, a countdown mechanism was added using "*react-countdown",* giving users a visible progress estimate during bulk AI processing:

```
<Countdown
  date={Date.now() + (length * 3000)}
  renderer={({ seconds, completed }) => {
    if (completed) setTimeoutReached(true);
    return <span> {seconds} seconds</span>;
  }}
/>
```

Figure 40: React component utilizing *react-countdown* to display a progress estimate during bulk AI processing for the AI-Tailored Flashcards feature.

This addressed the common performance bottleneck of sending dozens of cards to the API at once. By batching requests and introducing feedback via the countdown, the experience remained responsive and user-friendly even with large decks. A corresponding sketch of the review screen illustrates this logic in action – showing the card preview on the left, multiple suggestion buttons on the right, and a top-aligned countdown bar to indicate progress. This layout was designed to reduce visual clutter while emphasizing the most relevant decision points for the user. The overall architecture enabled both adaptability and transparency in how content was generated and reviewed, laying the foundation for further improvements in personalized language learning.

Building on this architecture, the language learning application generates dynamic tests to reinforce vocabulary acquisition and comprehension. Through AI-powered question generation, passage-based questions are designed to prompt users to fill in the blanks, with multiple-choice options provided to guide learning. This dynamic format allows for multiple variations and personalized experiences, ensuring that tests remain engaging and tailored to each learner's needs. The OpenAI API returns test data in the following format:

```
{
  "passage": {
    "text": "El ____ (1) zorro marrón salta sobre el ____ (2) perro perezoso.",
    "translation": "The ____ (1) brown fox jumps over the ____ (2) lazy dog.",
    "context": "This sentence is often used for typing practice and contains all
the letters of the alphabet. It's a popular pangram used to improve typing speed
and accuracy.",
    "vocab_hints": {
      "rápido": "quick",
      "zorro": "fox",
      "marrón": "brown",
      "perro": "dog",
      "perezoso": "lazy"
    },
    "blanks": {
      "1": {
        "question": "The ____ (1) brown fox.",
        "options": ["quick", "lazy", "heavy", "slow"],
        "correct_answer": "quick"
      },
      "2": {
        "question": "The ____ (2) lazy dog.",
        "options": ["quick", "lazy", "heavy", "slow"],
        "correct_answer": "lazy"
      }
    }
  }
}
```

Figure 41: Example JSON structure returned by the OpenAI API for generating the Testing Page passage, including the passage text with blank markers, correct words for blanks, and multiple-choice options.

As the application scales, it becomes important to ensure that the flashcard data used for test generation is manageable and efficient. Flashcards contain multiple pieces of information (e.g., *cardScore, engCard, userLangCard*), making them complex objects to handle in tests. To optimize performance and maintainability, the function *subsetCards()* is used to extract only the essential information needed for test generation. This simplifies the test creation process and ensures the app runs efficiently even with larger datasets.

```
export function subsetCards(cards: Card[]): TestCard[] {
  return cards.map(({ engCard, userLangCard }) => ({ engCard, userLangCard }));
}
```

Figure 42: Utility function *subsetCards* used to extract only essential fields (e.g., *engCard*, *userLangCard*) from full flashcard objects, optimizing data handling for test generation.

By limiting the data passed into functions such as *createTest()* and local generators, the system minimizes unnecessary data manipulation, improving both speed and clarity. This optimization ensures that the test generation process remains efficient, especially as the complexity of the tests increases. As a result, the application can handle large decks of flashcards without sacrificing performance or user experience.

Building upon this efficient data handling, the system employs utility functions to generate various types of questions, including synonym matching and word scramble questions. For example, the function for generating synonym matching questions works by selecting random flashcards, ensuring that each test is unique. It also adds distractor options to increase the challenge and randomness of the questions, further enhancing the learning experience.

```
export function generateSynonymMatchingQuestions(flashcards: TestCard[], count =
4): Question[] {
  return flashcards.slice(0, count).map(card => {
    const options = [card.engCard];
    // Add distractors
    while (options.length < 4) {
      const randomCard = flashcards[Math.floor(Math.random() * flashcards.lengt
h)];
      if (!options.includes(randomCard.engCard)) options.push(randomCard.engCar
d);
    }
    return {
      question: `The word **${card.userLangCard}** has the closest meaning to whi
ch word below?`,
      options: options.sort(() => 0.5 - Math.random()),
      correct_answer: card.engCard
    };
  });
}
```

Figure 43: Utility function *generateSynonymQuestions* creating multiple-choice synonym questions by selecting a correct card and adding random distractor options from the available vocabulary.

Similarly, the word scramble question generator shuffles the letters of words randomly, prompting users to unscramble them. This added variation introduces an additional layer of difficulty, requiring users to engage with vocabulary in a different way.

```
export function generateWordScrambleQuestions(flashcards: TestCard[], count: numb
er = 4): Question[] {
    const shuffled = flashcards.sort(() => 0.5 - Math.random())
    return shuffled.slice(0, count).map(card => {
        const scrambledWord = card.userLangCard
            .split('')
            .sort(() => 0.5 - Math.random())
            .join('')
        return {
            question: `Unscrambled this word **${scrambledWord}**`,
            correct_answer: card.userLangCard
        }
    })
}
```

Figure 44: Utility function *generateWordScrambleQuestions* creating word scramble challenges by shuffling the letters of vocabulary words.

Several performance bottlenecks and bugs were identified around question rendering, particularly with resetting states, radio button interactions, and page navigation. To address these issues, the RadioGroup component from *ShadcnUI* was used more effectively, controlling the state of user inputs and ensuring that answer selections are accurately tracked even after page navigation.

Additionally, dynamic rendering was implemented using "*Object.entries(passage.blanks).map(...)*" to render questions and answers based on the AI-generated test format. These optimizations allow for smooth transitions between questions and pages, providing a seamless user experience.

```
<RadioGroup
  disabled={isSubmitted}
  onValueChange={(value) => handleAnswer(`passage_${id}`, value)}
  value={answers[`passage_${id}`] || ""}
>
  {blank.options.map(option => (
    <div key={option}>
      <RadioGroupItem value={option} id={`${id}_${option}`} />
      <Label htmlFor={`${id}_${option}`}>{option}</Label>
    </div>
  ))}
</RadioGroup>
```

Figure 45: Frontend code snippet demonstrating dynamic rendering of fill-in-the-blank questions and radio button answer options using *Object.entries().map()* based on the AI-generated test data.

To enhance the learning experience, a *QuestionNav* component was added to allow users to visually track their progress across the questions. It provides real-time feedback by marking questions as answered and helps users easily navigate through the test. This feature improves engagement by enabling quick jumps to previously answered questions and highlighting completed sections.

```
<Button
  key={num}
  onClick={() => onQuestionClick(num)}
  className={`${isQuestionAnswered(num) ? 'bg-green-500 opacity-50' : 'bg-blue-50
0'} hover:opacity-100`}
>
  {num}. {isQuestionAnswered(num) && '✓'}
</Button>
```

Figure 46: React component *QuestionNav* providing visual navigation for the Testing Page, allowing users to see answered questions and jump between sections.

A major challenge faced during development was managing the Countdown timer. React's rendering behavior, which causes components to re-render when switching between sections, caused issues with the timer's state. To resolve this, a timer state was stored in a *useRef*, preventing unnecessary resets when the component re-renders. This solution ensures the countdown timer remains accurate, even when the component is re-rendered due to navigation or state changes.

```
const endTimeRef = useRef(endTime); // Store the end time
useEffect(() => {
  endTimeRef.current = endTime; // Update ref with new end time
}, [endTime]);

<p>Time left: <Countdown date={endTimeRef.current} renderer={renderer} /></p>
```

Figure 47: Code snippet showing the use of *useRef* to store and manage the countdown timer state, preventing resets during React component re-renders.

Despite the reuse of components from developed in Section 5.2.1 - which focused on integrating OpenAI's API for generating personalized vocabulary explanations – several implementation challenges emerged during the expansion of personalization features. While the prompt generation and dictionary scraping functions continued to operate reliably, maintaining semantic precision proved consistently difficult. The AI occasionally suggested inappropriate vocabulary replacements or incorrect contextual corrections, particularly in cases involving abstract terms or idiomatic expressions. This compromised user confidence, especially when dealing with unfamiliar vocabulary, even when dictionary definitions and explanations were provided alongside the AI-generated suggestions. The intended user-guided decision-making process, aimed at fostering deeper

engagement, often became tedious and slow, thereby conflicting with the application's core objective of promoting efficient flashcard-based learning.

To address these issues, prompt templates were iteratively refined, and semantic validation layers were introduced. One experimental approach involved assessing the similarity between the original and suggested vocabulary using OpenAI's embedding API, followed by cosine similarity checks. The embedding API convert words or phrases into high-dimensional numerical vectors that capture their semantic meaning based on usage in large text corpora. By comparing these vectors, it becomes possible to evaluate how closely related two terms are in meaning, rather than just form. Suggestions falling below a semantic similarity threshold were either deprioritized or excluded from the final output. The following simplified TypeScript example demonstrates how the filtering logic was implemented using OpenAI's embedding API and the *cosine-similarity* function:

```
import axios from 'axios';
import cosineSimilarity from 'cosine-similarity';

const getEmbedding = async (text: string): Promise<number[]> => {
  const { data } = await axios.post(
    'https://api.openai.com/v1/embeddings',
    { input: text, model: 'text-embedding-ada-002' },
    { headers: { 'Authorization': `Bearer ${process.env.OPENAI_API_KEY}` } }
  );
  return data.data[0].embedding;
};

const isSemanticallySimilar = async (word1: string, word2: string) => {
  const [embed1, embed2] = await Promise.all([getEmbedding(word1), getEmbedding(w
ord2)]);
  return cosineSimilarity(embed1, embed2) >= 0.4;
};

isSemanticallySimilar('resilient', 'bouncy').then(similar => {
  console.log(similar ? "Accepted" : "Rejected");
});
```

Figure 48: Simplified TypeScript example demonstrating the experimental use of OpenAI's embedding API and cosine-similarity to filter AI-suggested flashcard refinements based on semantic similarity to the original term.

While still in the evaluation stage, this method showed early promise in reducing semantically irrelevant suggestions. However, ambiguity persisted in edge cases where all available options – dictionary-scraped synonyms, AI-generated suggestions, or retaining the original term – failed to match the intended context. A potential future solution involves constructing a small, labeled dataset derived from user interactions, where rejected suggestions are logged and categorized. Over time, this could serve as training data for fine-tuning a lightweight model or filtering layer, enhancing future suggestions through context-aware feedback loops. This approach, however, requires a sufficient volume of consistent user input and raises new challenges regarding data quality and privacy, which remain outside the current scope.

Additionally, the dictionary scraping feature, though effective, raises sustainability and compliance concerns. While no immediate issues were encountered during development, long-term use may require transitioning to an official licensing model by purchasing access to a commercial dictionary API. This would ensure stable performance and legal clarity while enabling more advanced querying options (e.g., frequency data, part-of-speech filtering, and example-rich definitions).

On the Testing Page, initial test generation methods led to incoherent output. A typical failure case involved selecting ten random flashcards without semantic relation, prompting the AI to generate disjointed or nonsensical passages. This issues was addressed by allowing the AI to filter and select only compatible vocabulary subsets. The following code excerpt illustrates the adjusted logic within the *createTest()* utility function:

```
const validTerms = flashcards.filter(
  card => card.cardScore > 0 && card.engCard.length > 2
);
const prompt = generatePassagePrompt(validTerms.slice(0, 7)); // capped at 7 for
coherence
```

Figure 49: Code excerpt from the *createTest* utility function illustrating the adjusted logic where the AI selects a compatible subset of vocabulary (*compatibleVocabulary*) before generating the test passage, improving coherence.

Planned improvements for the Testing Page include limiting test generation to smaller, high-relevance vocabulary sets to reduce progressing time and improve coherence. Smarter filtering algorithms based on user performance metrics – such as correctness rate and vocabulary complexity – will be used to balance question difficulty across sections. To enhance user experience, UI-level adjustments such as persistent timers, clearer progress indicators, and smoother navigation will be prioritized. Additional test types, such as another type of fill-in-the-blank or contextual matching, are also under consideration, depending on future usability testing and technical feasibility.

These developments mark a significant step toward a more personalized, efficient, and engaging language learning experience. By combining generative AI, interactive UI components, and thoughtful data optimization, the application delivers tailored flashcard reviews and adaptive testing that respond to individual learner needs. While challenges remain – particularly around semantic precision and long-term sustainability of third-party tools – the foundation is in place for continued innovation. Future improvements may include fine-tuned AI models based on user feedback, officially licensed dictionary integrations, and more nuanced difficulty adjustments. Together, these personalization features reflect a broader commitment to scalable user-centric language learning powered by modern AI.

5.2.5 Additional Features

In addition to the core learning functionalities, several supporting features were implemented to enhance the overall user experience. These include a login/register system, a dictionary page, and a duplicate-checking tool. Each of those features plays a key role in improving usability, personalization, and long-term engagement. Beginning with the login and registration flow, this feature forms the foundation of user-specific training, progression tracking, and gamification.

The login and registration functionality are kept simple and accessible. To register, users provide a name, username, and a 4-digit PIN, which acts like a short password or OTP.

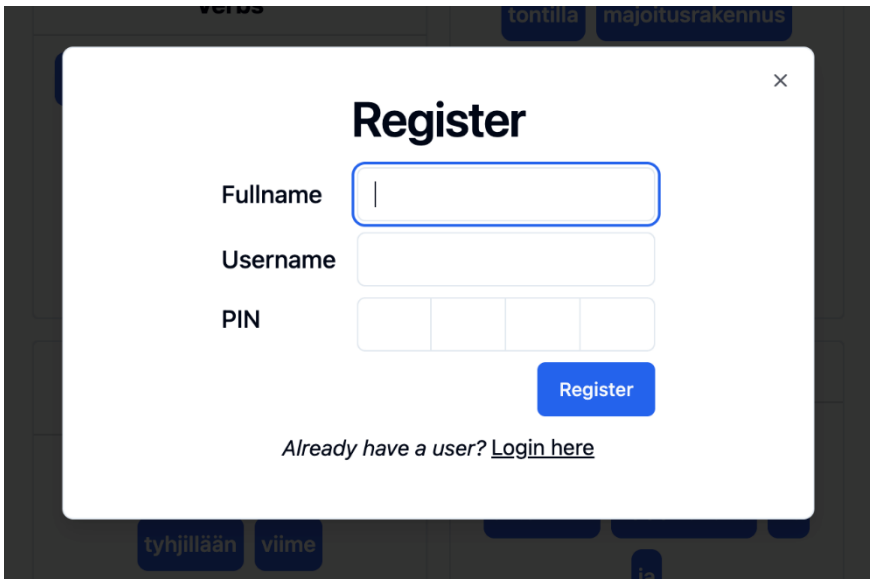


Figure 50: Registration form interface showing fields for Fullname, Username, and PIN.

The login process requires just the username and this PIN.

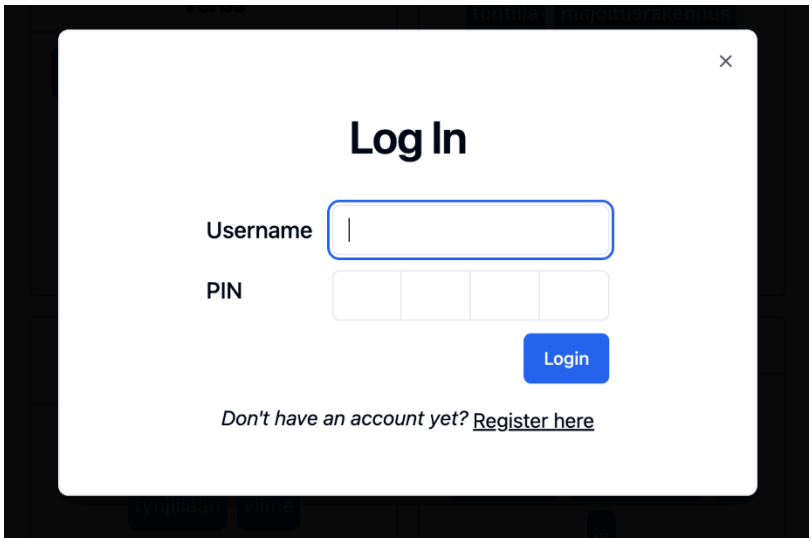


Figure 51: Login form interface showing fields for Username and PIN.

Each user is stored in a MongoDB schema with associated data like XP, levels, streaks, badges, weekly progress, and region information. Although the authentication itself is straightforward, it allows the system to support user-specific decks and learning history.

```
const UserSchema = new Schema({
  username: { type: String, required: true, unique: true },
  pin: { type: String, required: true },
  level: { type: Number, default: 1 },
  xp: { type: Number, default: 0 },
  region: { type: String, default: null }
})
```

Figure 52: Snippet of the MongoDB User schema (likely via a tool like MongoDB Compass or similar) showing stored user data fields like xp, level, *currentStreak*, badges, etc..

This feature was prioritized early, as saving flashcards, progress, and gamified rewards all require a persistent identity. Users can still perform lightweight tasks like translating a phrase without logging in, but to access quizzes or interactive learning experiences, authentication is necessary. While there are no specific academic sources referenced, this approach aligns with standard practices in user-centered educational tools where personalization boots motivation and retention.

The login system was built using the MERN stack, leveraging Express.js and MongoDB on the backend. Upon registration, a user's IP is processed using *geoip-lite* to estimate their region, which is then saved for XP leaderboards and analytics.

Passwords are hashed with *bcrypt,* and *JWTs* are used for session management via *HTTP-only cookies*. The API handles registration, login, logout, profile fetching, and user updates. If the region isn't known during login or profile access, it is re-fetched based on the user's IP. This ensures regional data remains consistent even if it wasn't available during the initial registration. Here's how password hashing is handled during registration:

```
const salt = bcrypt.genSaltSync(10)
const hashedPin = bcrypt.hashSync(pin, salt)
const region = getRegionFromIP(req.ip)
const createdUser = await User.create({ username, pin: hashedPin, region })
```

Figure 53: Backend code snippet showing the use of *bcrypt.hash* to securely hash the user's PIN during registration before saving it to the database.

During login, the server compares the stored hash and, if successful, issues a JWT token stored in cookies:

```
const user = await User.findOne({ username })
const isPasswordCorrect = user && bcrypt.compareSync(pin, user.pin)
if (!isPasswordCorrect) return res.status(422).json('Invalid credentials')

jwt.sign({ id: user._id }, JWT_SECRET, {}, (err, token) => {
  if (err) throw err
  res.cookie('token', token, { httpOnly: true }).json(user)
})
```

Figure 54: Backend code snippet from the login route showing the comparison of the provided PIN with the stored hash using *bcrypt.compare* and the subsequent generation of a JWT upon successful authentication.

The development process went smoothly overall, though some early challenges were encountered – particularly around user authentication and form validation. Initially, users could access other users' data due to missing server-side verification. This was later addressed by introducing a JWT verification middleware (*verifyToken)* to protect endpoints and ensure that users can only access their own resources.

```
const verifyToken = (req, res, next) => {
    const { token } = req.cookies;
    if (!token) {
        return res.status(401).json({ error: 'Unauthorized' })
    }
    try {
        const userData = jwt.verify(token, JWT_SECRET)
        req.userData = userData
        next()
    } catch (error) {
        console.error('Error verifying token: ', error)
        return res.status(401).json({ error: 'Unauthorized' })
    }
}
```

Figure 55: *verifyToken* middleware function used to secure private routes.

To improve the front-end experience, usability issues were also tackled on the login and registration pages. One problem was that clicking "Choose deck" without being logged in didn't trigger any response, which created confusion. A modal was added to automatically prompt users to log in or register when unauthenticated users attempted to interact with protected features.

```
<Dialog>
  <DialogTrigger>
      <Button type="submit" className="w-full sm:w-auto">Save this to a deck</Button>
  </DialogTrigger>
    <DialogContent className={user ? "p-0 flex gap-4 flex-grow-1" : ""}>
      {user ? (
        <DisplayCurrentDecks onSelectDeck={saveWordToDeck} />) : (<>
      <DialogTitle className="text-2xl sm:text-4xl flex items-center justify-center mt-8">Log In</DialogTitle>
      <LoginPage /></>)}
      {openNewDeck && <NewDeckCard setOpenNewDeck={setOpenNewDeck} />}
    </DialogContent>
</Dialog>
```

Figure 56: Screenshot of the modal prompting users to log in or register when attempting to access a protected feature while unauthenticated.

The original implementation also relied on props to pass user information between components, which complicated state management. This was refactored so that both the login and register modals could handle their own form logic, validation, and submission independently. For example, in the updated Register form, all validation checks are handled locally, and the user receives instant feedback when issues occur. These improvements made the authentication process more intuitive and reliable.

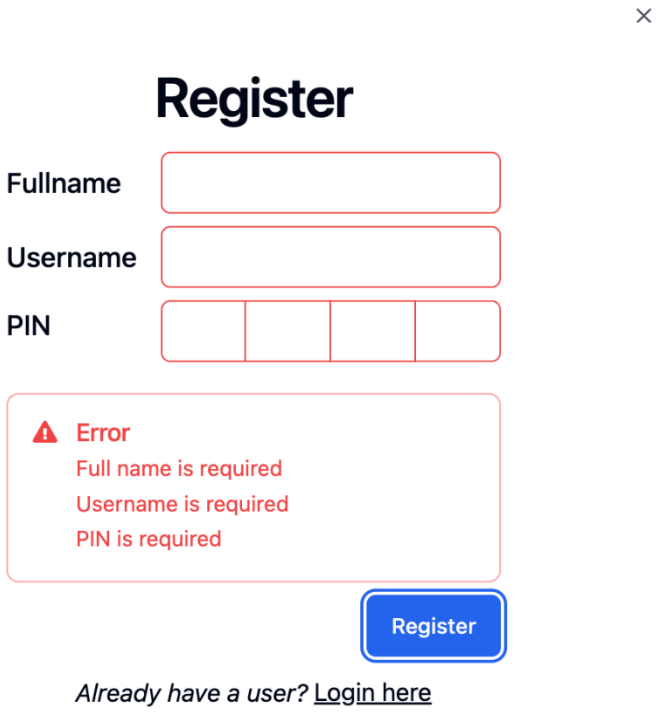


Figure 57: Register component validation ensures all fields are completed and valid before submitting

There are still areas for improvement. For example, social login options like Google or Apple could simplify the sign-up flow. The current implementation also lacks password confirmation during registration, making it easy to mistype the PIN. While the 4-digit format helps keep the login fast, it may be too simple from a security perspective. Shifting to longer passwords in the future could address this, though it might require rethinking the UI. Lastly, the PIN input isn't currently masked

during typing, making it visible on-screen – this was a limitation of the library used and might be addressed by switching to another input solution later.

To address another usability pain point, the application also includes a Dictionary Page – a feature designed to improve how users interact with their stored vocabulary. As the number of flashcards grows, especially across multiple decks, finding a specific word can become time-consuming. The Dictionary Page solves this by consolidating all saved terms and definitions into a single searchable view.

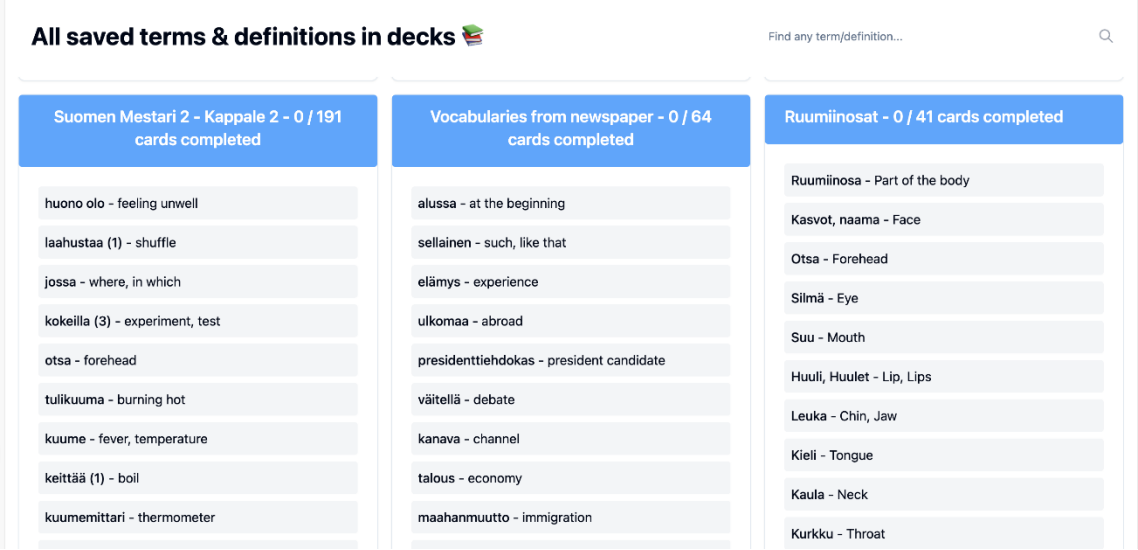


Figure 58: Screenshot of the Dictionary Page interface, showing a search bar and consolidated vocabulary entries from user decks.

This feature was motivated by personal experience. After adding hundreds or even thousands of words into various decks, it became increasingly difficult to remember where a word was stored or what it meant without reviewing multiple decks manually. In situations requiring a quick lookup – such as during conversation practice or reading – this delay proved frustrating. The Dictionary Page was built to eliminate that friction.

Technically, the page uses a simple but effective search mechanism. Users can search for terms and definitions in either the target language or their native language. This is implemented by filtering the *decks* context based on a debounced search term:

```
const newFilterDecks = decks.map(deck => ({
    ...deck,
    cards: deck.cards.filter(card =>
        card.engCard.toLowerCase().includes(debouncedSearchTerm.toLowerCase()) |
|
        card.userLangCard.toLowerCase().includes(debouncedSearchTerm.toLowerCase
())
    )
})).filter(deck => deck.cards.length > 0)
```

Figure 59: Frontend code snippet showing the filtering logic for the Dictionary Page search, comparing the search term against *userLangCard* and *engCard*.

To enhance performance and avoid lag during fast typing, the search input is debounced using the *useDebounce* hook from the *@uidotdev/usehooks* library. This introduces 500-millisecond delay before triggering the filtering logic, ensuring smoother input handling even with large datasets. The debounced search term is derived as follows:

```
const debouncedSearchTerm = useDebounce(searchTerm, 500)
```

Figure 60: Frontend code utilizing the *useDebounce* hook to delay the execution of the dictionary search filtering, improving performance during typing.

The page structured displays each deck as a card, showing its name and the number of completed flashcards. This completion ratio is calculated dynamically and rendered directly within the UI. Within each deck card, individual vocabulary items are listed in a loop. Cards that have a perfect score (i.e., fully learned) are highlighted in green using conditional styling. In addition, matching search terms are emphasized through yellow background highlighting, improving readability when filtering results. Both behaviors are implemented through simple but effective logic in the UI code.

```
<ScrollArea className="h-[calc(100vh-8rem)]">
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
        {filteredDecks.map((deck) => (
            <Card key={deck._id} className="flex flex-col">
                <CardHeader className="bg-blue-400 rounded py-3">
                    <h2 className="text-xl font-semibold mb-2 text-white text-center items-center flex">
                        {deck.deckName} - {deck.cards.filter((card) => card.cardScore == 5).length} /
                        {deck.cards.length} cards completed
                    </h2>
                </CardHeader>
                <CardContent className="flex-grow p-4">
                    {deck.cards.map((card) => (
                        <div key={card._id}
                            className={`p-2 m-2 rounded ${card.cardScore === 5 ? 'bg-green-100' : 'bg-gray-100'}`}
                        >
                            <p>
                            <span className="font-medium">
                            {highlightText(card.userLangCard)}
                            </span> - {highlightText(card.engCard)}</p>
                        </div>
                    ))}
                </CardContent>
            </Card>
        ))}
    </div>
</ScrollArea>
```

Figure 61: Code snippet that renders deck cards, conditionally styles completed flashcards in green, and highlights matching text in yellow.

This logic produces a clean and responsive layout where vocabulary items with a perfect score are easily distinguishable in green, and matching terms (e.g., “balliballi”) are visually highlighted in yellow to improve scanning. The layout uses a scrollable container and a responsive grid to ensure usability across screen sizes.

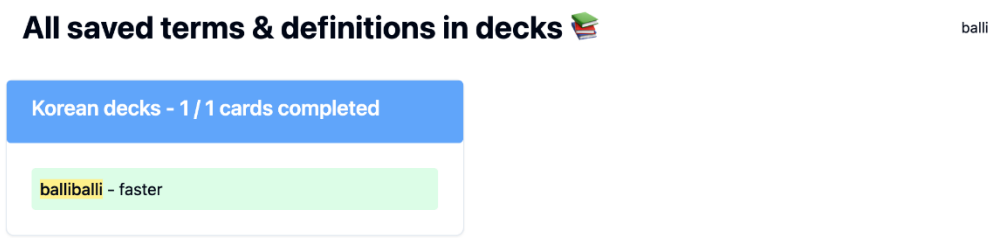


Figure 62: Live example of the dictionary UI showing green-highlighted completed cards and yellow-highlighted matching search term “balliballi”

Built using the MERN stack and styled with Shadcn/UI, the design prioritizes clarity and accessibility. Scroll containers and input fields are customized for smooth interaction, while the overall interface maintains consistency with the rest of the app.

There were no major implementation challenges for this feature. However, potential improvements include adding filters and sorting options, allowing users to tag words with labels, supporting CSV

export of vocabulary, or integrating speech synthesis for pronunciation. These enhancements could further elevate the feature's utility and make it more interactive and learner-friendly.

Another crucial supporting feature implemented to maintain data integrity and enhance usability is the duplicate-checking tool. This tool proactively identifies potential duplicate vocabulary entries while a user is manually adding or editing cards within a deck. It checks if the term being added already exists, either within the current deck or across all decks associated with the user's account. Upon detection, the interface presents a warning and offers several options for resolution: delete the new card, delete the existing duplicates found in other decks, keep the new card and suppress future warnings for it, keep all duplicates, or restrict the check to only the current deck. This functionality directly addresses the problem of redundant vocabulary entries, which could otherwise compromise the effectiveness of the spaced repetition system. Learning the same term multiple times from different card entries can disrupt the calculated review intervals and lead to an inefficient learning experience by presenting already known material too often or resetting progress unnecessarily. Therefore, the duplicate finding tool is essential for preserving the integrity of the learning data and ensuring the spaced repetition algorithm functions optimally.

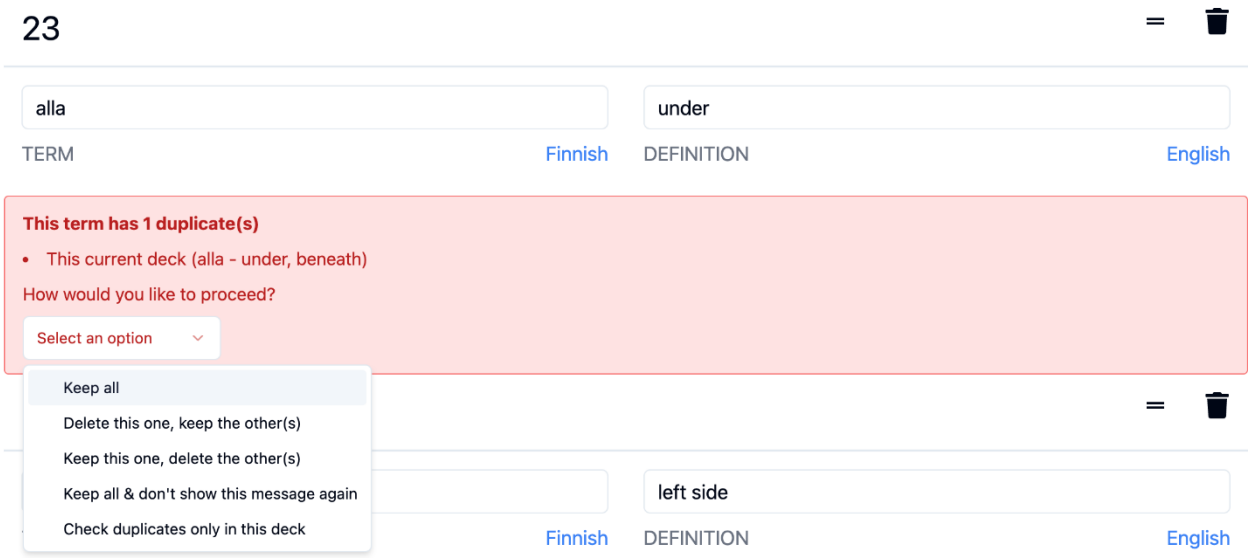


Figure 63: Interface warning displayed when the duplicate-checking tool detects a potential duplicate vocabulary entry during card editing.

The design approach for this feature centers around a custom React hook, *useFindDuplicates*, which isolates the logic for checking terms across the user's decks. The core of the detection mechanism within this hook involves iterating through decks and filtering cards based on matching terms (*userLangCard*), while excluding the card currently being checked (*c._id !== card._id*)

```
// useFindDuplicates.ts - Core duplicate finding logic within findDuplicatesForCard function
const duplicateCards = deck.cards.filter((c) =>
  (c.userLangCard === card.userLangCard) && (c._id !== card._id)
);
duplicateCards.forEach(duplicateCard => {
  duplicatesFound.push({
    deckName: deck.deckName === deckName ? 'This current deck' : `${deck.deckName}`,
    isDuplicateTerm: true, // Simplified for brevity
    definition: duplicateCard.engCard,
    cardId: duplicateCard._id
  });
});
```

Figure 64: Core detection logic within the useFindDuplicates hook, iterating through decks and filtering cards based on matching terms while excluding the current card.

In the user interface, specifically within the *EditCardDetails* component, the results from this hook trigger a conditional warning. A key part of this is the *renderDuplicateWarning* function, which constructs the alert message and presents the user with actionable choices via a dropdown menu (*Select* component)

```
// EditCardDetails.tsx - Snippet from renderDuplicateWarning function
const renderDuplicateWarning = (card: Card, cardDuplicates: any[]) => (
  <div className="bg-red-100 border border-red-400 text-red-700 px-4 py-3 rounded relative mt-2" role="alert">
    <strong className="font-bold">
      This term "{card.userLangCard}" has {cardDuplicates.length} duplicate(s) found in:
    </strong>
    <ul className="mt-2 list-disc list-inside">
      {/* List details of each duplicate (mapping logic omitted for brevity) */}
    </ul>
    <p className="mt-2">How would you like to proceed?</p>
    {/* Dropdown for user actions using ShadcnUI Select component */}
    <Select onValueChange={(value) => handleDuplicateAction(card._id, value, cardDuplicates.map(d => d.cardId))}>
      <SelectTrigger> {/* Content omitted for brevity */} </SelectTrigger>
      <SelectContent>
        <SelectItem value="KEEP-ALL">Keep this card and ignore this warning</SelectItem>
        <SelectItem value="DELETE-THIS">Delete this card, keep the existing one(s)</SelectItem>
        <SelectItem value="DELETE-OTHER">Keep this card, delete the existing one(s)</SelectItem>
        <SelectItem value="KEEP-NO-SHOW">Keep all duplicates & don't show warnings again</SelectItem>
        <SelectItem value="CHECK-ONLY-CURRENT">Check duplicates only in this deck</SelectItem>
      </SelectContent>
    </Select>
  </div>
);
```

Figure 65: Frontend component *renderDuplicateWarning* constructing the alert message and dropdown menu (Select component) presenting resolution options to the user when duplicates are found.

The user's choice from the dropdown invokes the *handleDuplicateAction* function. This function uses a *switch* statement to manage the different resolution paths, interacting with component state and the *useFindDuplicates* hook's functions (*updateLocalDecks, setDuplicates)* to modify card data or warning visibility accordingly. For instance, handling the deletion of other duplicates involves filtering the *localDecks* state managed by the hook.

```
// EditCardDetails.tsx - Snippet from handleDuplicateAction switch case
const handleDuplicateAction = (cardId: string, action: string, duplicateCardIds?: string[]) => {
    switch (action) {
        case 'DELETE-THIS': {
            // Logic to find and delete the current card (details omitted)
            break;
        }
        case 'DELETE-OTHER':
            // Update localDecks by filtering out cards matching duplicate IDs
            if (duplicateCardIds && duplicateCardIds.length > 0) {
              updateLocalDecks(localDecks.map(deck => ({
                  ...deck,
                  cards: deck.cards.filter(c => !duplicateCardIds.includes(c._id))
              })));
              // Remove the warning for this card ID (state update logic omitted)
            }
            break;
        // Other cases (KEEP-NO-SHOW, KEEP-ALL, CHECK-ONLY-CURRENT) modify state flags
        // or filter the duplicates state (details omitted)
    }
  };
```

Figure 66: The *handleDuplicateAction* function using a switch statement to manage different user choices for resolving detected duplicates (e.g., deleting, keeping).

This structure ensures that duplicate detection logic is encapsulated, the UI provides clear warnings and options, and user actions are handled cleanly to maintain data consistency during the card editing process.

The development of the duplication detection feature involved addressing several non-trivial challenges related to data consistency, interface feedback, and edge case handling. One recurring issue was the persistence of false-positive duplicate warnings. This behavior was caused by a reliance on context-provided deck data, which was not updated immediately after local changes. Since the duplication check operated on outdated context data, removed duplicates continued to be flagged. To resolve this, the implementation was modified to utilize a locally maintained copy of the decks, ensuring that duplicate checks were always performed against the most recent state.

Another issue involved the unexpected appearance of multiple duplicate warnings after a single user action. For example, selecting the “delete” option for one duplicate triggered re-evaluations that surfaced additional warnings, even for unaffected cards. This indicated flaws in how warnings were propagated and cleared in response to state updates. Additionally, inconsistent behavior was observed in the “delete-other” option, where associated duplicates were not always removed as intended. These issues were documented and tracked systematically, with detailed observations recorded in GitHub Issue #41. The following figure provides a snapshot of the recorded debugging process:

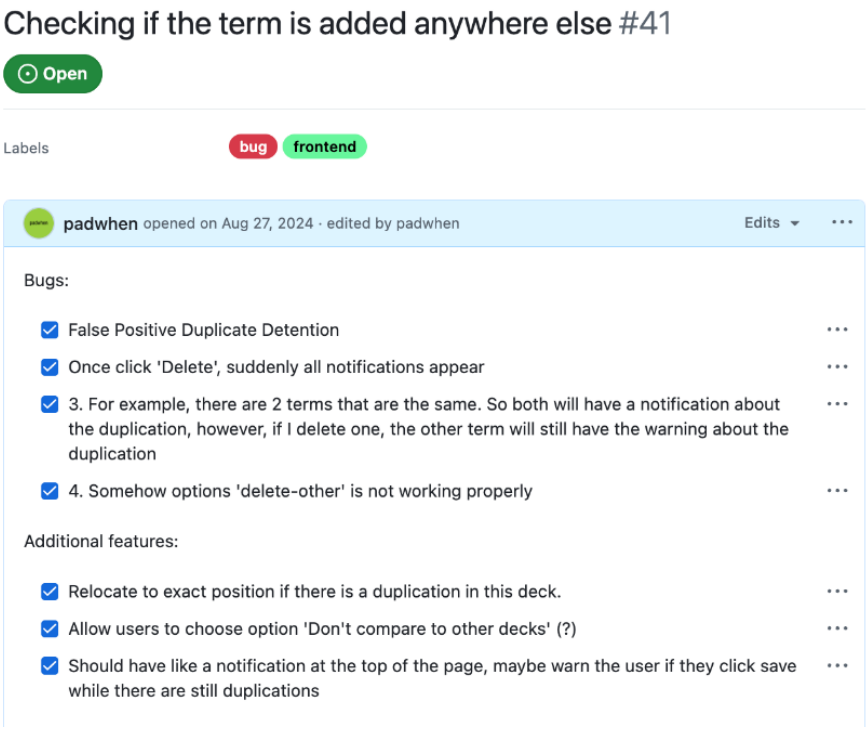


Figure 67: Screenshot of GitHub Issue #41 documenting bugs and observations related to the duplicate detection feature during debugging.

The process of uncovering these bugs was guided by continuous testing and critical evaluation from the user's perspective. By simulating realistic user interactions – such as editing multiple cards with similar terms and switching decks mid-session – the feature's weaknesses became more apparent. These findings directly informed further refinements, including optional exclusion of other decks during duplicate checks, visual warnings during the save process, and scroll-based navigation to the duplicate's location within the same deck.

There remains room for improvement in the duplicate detection system, particularly in enhancing the user experience and system performance for larger datasets. While the current solution provides real-time detection and resolution options, future iterations could benefit from asynchronous background checks to reduce UI interruption. Additionally, integrating fuzzy matching or typo tolerance could expand the system's ability to catch near-duplicates that currently pass undetected. More intuitive UI cues, such as inline highlights or preview panels of detected duplicates, could also improve clarity during the editing process.

Together, these supplementary features contribute to a more robust and user centered application. While not part of the core training flow, they plan an essential role in supporting and enriching the learning process. By enabling account-based tracking, reducing friction during card management, and preventing redundant entries, these features help maintain data integrity and improve the long-term efficiency of the system. Their inclusion reflects a broader design philosophy focused on sustainability, personalization, and overall learning quality.

5.3 Testing and Refinement

After the core functionalities were implemented, a focused period of testing and refinement was conducted to ensure application stability and usability. This phase involved validating both the technical correctness of the components and the overall user experience. The aim was to identify

and resolve issues related to performance, edge cases, and interaction logic across different features. Feedback from actual usage scenarios and continuous internal evaluations informed necessary adjustments and enhancements, leading to a more polished and reliable final product.

5.3.1 Testing

Testing played a crucial role throughout the development of the application to ensure reliability, prevent regressions, and improve user experience. This section describes the different testing methodologies used, including unit tests, end-to-end (E2E) tests, and user testing helped automate and enforce quality checks.

Unit testing in this application was centered on verifying the behavior of small, isolated logic units such as utility functions and custom React hooks. The tests were written primarily using Vitest alongside React Testing Library. Key areas included hooks like *useFindDuplicates* hook was tested to ensure it accurately detects duplicate cards across different decks, updating its internal state when local decks or language tags change. These utilized mock data and simulated various deck configurations. One of the assertions looked like this:

```
expect(result.current.duplicates).toEqual({
  '1': [{ deckName: 'This current deck', ... }],
  '2': [{ deckName: '"French Advanced"', ... }]
})
```

Figure 68: Example Vitest unit test assertion for the *useFindDuplicates* hook, checking if the hook correctly identifies duplicates in a mock scenario.

Another hook, *useImportCards*, was tested with mocked dependencies using *vi.mock()* to isolate its logic from external modules like *parseImportData* data *uuid.v4*. For example:

```
vi.mock('@/utils/parseImportData')
vi.mock('uuid', () => ({ v4: () => 'mocked-uuid' }))
```

Figure 69: Example Vitest unit test setup for the *useImportCards* hook, using *vi.mock()* to mock dependencies and isolate the hook's logic for testing.

This approach allowed tests to focus on the hook’s internal logic – handling parsed data, changing separators, updating card state, and closing the modal upon successful import – without interference from the behavior of third-party utilities.

End-to-end (E2E) testing played a crucial role in validating the application’s critical user flows and ensuring that different parts of the system worked together seamlessly, from the user interface through the backend and database. Cypress was utilized as the primary tool for implementing these tests. The purpose of employing E2E tests was to simulate real user interactions, verifying core functionalities such as authentication, navigation, and the behavior of learning features under realistic conditions. These tests aimed to confirm that buttons triggered the correct actions,

interactive elements responded as expected, and features like keyboard navigation were functional across the application.

Authentication flows, including user login, were a key area covered by E2E tests to ensure users could securely access their accounts and data. The testing process involved navigating to the login page, entering credentials, and confirming successful login before proceeding to test protected routes and user-specific features.

```
beforeEach(function() {
    cy.visit('https://padwhen-learningapp.fly.dev')
    cy.viewport(1000, 660)
    cy.contains('Log In').click()
    cy.get('#username').type('1111')
    cy.get('[data-testid="pin-input"]')
    .find('input')
    .each(($el) => {
      cy.wrap($el).type('{selectall}') .type('1111')
    });
    cy.contains('Login').click();
})
```

Figure 70: Cypress E2E test code snippet demonstrating the simulation of a user login flow.

Navigation was another critical area subjected to E2E testing. Tests confirmed that users could navigate between different pages and sections of the application using both standard clickable links and keyboard shortcuts. This included verifying that header navigation worked correctly on different screen sizes and that breadcrumbs accurately reflected the user's current location within the application.

```
it('keyboard navigation works', () => {
    // accessToDeckDetails function is called here as a prerequisite
    cy.wait(500)
    cy.get('body').type('{rightarrow}')
    cy.get('[role="progressbar"] > div').should('have.attr', 'style', 'transform: translateX(-75%);');
    cy.get('[data-testid="current-card-number"]').should('contain.text', '2 /')
    cy.get('body').type('{leftarrow}')
    cy.get('[role="progressbar"] > div').should('have.attr', 'style', 'transform: translateX(-87.5%);');
    cy.get('[data-testid="current-card-number"]').should('contain.text', '1 /')
})
```

Figure 71: Cypress E2E test code snippet verifying navigation between pages and asserting the correct URL or breadcrumb display.

Testing extended to key learning features, such as interacting with flashcard decks and the learning page. E2E tests verified that users could access deck details, navigate through flashcards using UI elements and keyboard commands, and that dynamic elements like the progress bar updated correctly based on user interaction. Edge cases were also considered, including testing navigation behavior in decks with a single card or only two cards to ensure boundary conditions were handled appropriately. Tests also confirmed the conditional visibility and behavior of elements like

the “Learn” link, ensuring it was clickable only when a deck met the minimum card requirement for a learning session or displayed a tooltip otherwise.

```
it('progress bar moves on card navigation', function() {
    // accessToDeckDetails function is called here as a prerequisite
    cy.get('[data-testid="current-card-number"]').should('contain.text', '1 /')
    cy.get('[role="progressbar"]').should('have.attr', 'aria-valuemin', '0');
    cy.get('[role="progressbar"]').should('have.attr', 'aria-valuemax', '100');
    cy.get('[role="progressbar"] > div').should('have.attr', 'style', 'transform: translateX(-87.5%);');
    cy.get('[data-testid="move-right"]').click();
    cy.get('[data-testid="current-card-number"]').should('contain.text', '2 /')
    cy.get('[role="progressbar"] > div').should('have.attr', 'style', 'transform: translateX(-75%);');
    cy.get('[data-testid="move-left"]').click()
    cy.get('[data-testid="current-card-number"]').should('contain.text', '1 /')
    cy.get('[role="progressbar"] > div').should('have.attr', 'style', 'transform: translateX(-87.5%);');
})
```

Figure 72: Cypress E2E test code interacting with flashcard elements on the Deck Details page, checking navigation and dynamic element updates.

Tests for the learning page validated the interactive quiz elements and options. This included verifying that users could open the options dialog, toggle settings like including completed cards or shuffling, modify the quantity of quizzes, and select answer choices using both mouse clicks and number keys.

```
it('can select answers using number keys 1, 2, 3, 4 and by clicking', () => {
    const checkAnswerSelection = () => {
        cy.get('[data-testid="question-box"]')
            .find('[data-testid="answer-test"]')
            .should('have.length', 4)
            .and('exist');
        cy.wait(1500);
    }
    ['1','2','3','4'].forEach(key => {
        cy.get('body').type(key)
        checkAnswerSelection()
    })
})
```

Figure 73: Cypress E2E test code validating interactive elements on the Learning Page, such as toggling options or selecting quiz answers.

Reusable helper functions were implemented to streamline test creation and maintain adherence to the DRY (Don’t Repeat Yourself) principle (Wikipedia, 2025). The consistent use of *data-testid* attributes within the application’s HTML elements was a deliberate choice to ensure that tests remained robust and stable, unaffected by changes in CSS styling or minor alterations to the page layout. The testing approach also incorporated checking the application’s responsiveness by utilizing different viewport sizes, confirming that the user interface remained functional and accessible on both desktop and mobile dimensions. Verifying the correct navigation paths by asserting the URL after crucial clicks was also a standard practice in the E2E test suite. Currently, there are approximately 60 end-to-end tests implemented, and it is a mandatory step in the development workflow to ensure all these tests successfully pass before proceeding to deployment.

User testing in the early stages of development was primarily conducted by the developer, serving as both the creator and a key user of the application. This personal engagement provided

immediate insights into the application's usability and the effectiveness of its features from a learner's perspective. The development process was significantly shaped by this self-testing, leading to an ongoing internal dialogue about feature prioritization – balancing technical implementation with genuine user benefit. Over time, a small group of users also used the application, offering additional perspectives. Subsequently, the application was used by approximately 30 users, contributing to a broader understanding of user interaction and needs.

The feedback loop in these initial phases was largely informal, stemming directly from the developer's user experience and observations, supplemented by informal input from a small group of users. This approach allowed for rapid identification of pain points and opportunities for improvement. For instance, the personal struggle with remembering whether a specific vocabulary card had already been added to a deck directly inspired the development of the duplicate-checking tool, illustrating how user challenges informed specific feature implementations. While a structured survey was later conducted to gather broader feedback for the thesis evaluation, the refinement process during development was heavily influenced by these direct, user-centric insights.

5.3.2 Refinement

Refinement of the application was an ongoing process, driven by issues identified during development and testing. This iterative approach ensured that feedback from testing cycles directly informed improvements to the application's functionality, usability, and performance. Key refinements addressed technical challenges and enhanced the user experience based on observed behavior and reported issues.

Specific refinements included standardizing API interactions. Inconsistent data formatting from external services, revealed during testing, was resolved by implementing a strict schema within API prompts, ensuring predictable and usable responses. Addressing user interface inconsistencies across different screen sizes was also necessary. Testing on various viewports highlighted layout issues that were corrected by moving away from fixed sizing to an adaptive design approach.

Functionality fixes resulting from testing included making interactive features fully accessible and operational. The Matching Game and Flashcard Page, initially difficult to navigate to, had their routing corrected. The "Get a Hint" feature on the flashcard page and data management options like deleting decks and exporting to CSV were implemented or fixed after testing showed they were non-functional.

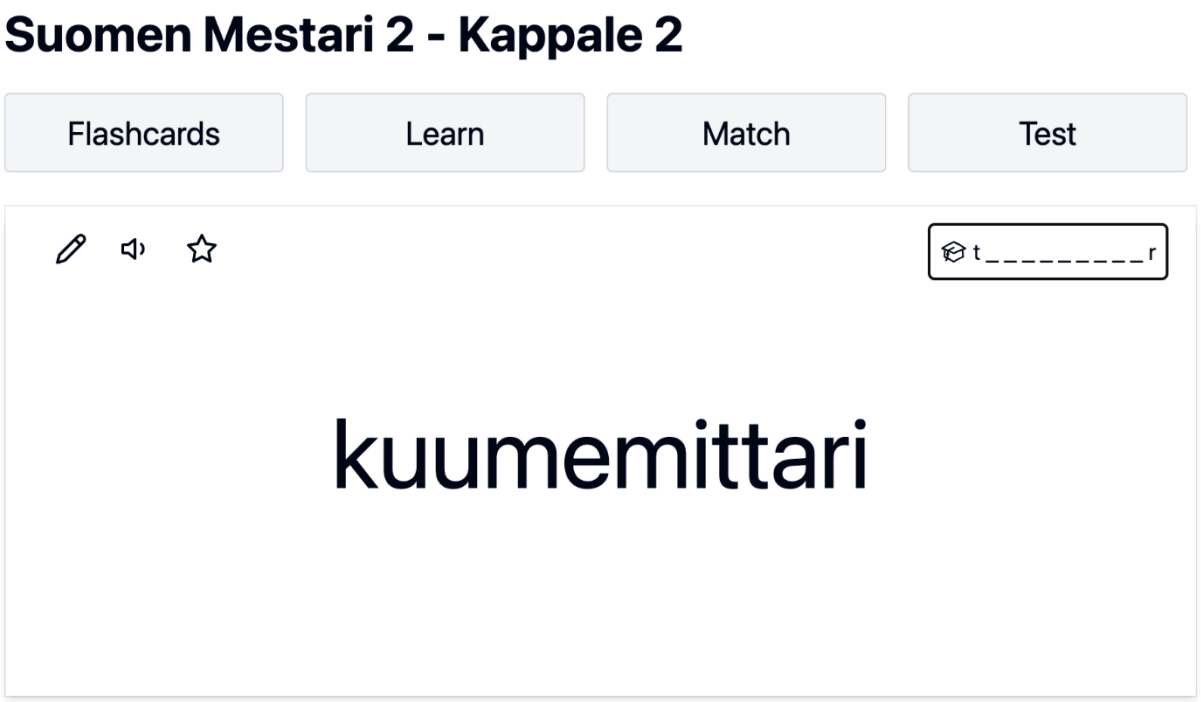


Figure 74: Screenshot illustrating UI refinements made based on testing feedback.

The development process also saw refinements inspired by direct user experience, particularly the developer's own interaction with the application. The frequent uncertainty about whether a vocabulary card had already been created led to the development of the duplicate-checking tool, refining the user's ability to manage their vocabulary efficiently. These examples illustrate how testing insights translated into concrete improvements, enhancing the application's reliability and usability throughout development.

5.4 Deployment

Deployment of the application utilized Continuous Integration/Continuous Deployment (CI/CD) pipelines to automate the process of building, testing, and deploying code changes. This approach ensured that updates were integrated and delivered efficiently and reliably. GitHub Actions was used to orchestrate these pipelines, defining automated workflows that triggered upon specific events, such as pushing code to the main branch or opening a pull request.

Two primary pipeline configurations were instrumental in the deployment strategy. The *healthcheck.yml* pipeline was configured to perform regular checks on the deployed service URL.

```yaml
name: Health Check

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
    types:
      - opened
      - synchronize
  schedule:
    - cron: "30 12 * * *"

jobs:
  health-check:
    runs-on: ubuntu-20.04

    steps:
      - name: Check the deployed service URL
        uses: jtalk/url-health-check-action@v4
        with:
          url: https://padwhen-learningapp.fly.dev/
          max-attempts: 3
          retry-delay: 5s
```

Figure 75: GitHub Actions workflow definition (*healthcheck.yml*) configured to perform regular health checks on the deployed application URL.

This health check pipeline ran on pushes, pull requests, and a scheduled *cron* job, providing automated monitoring of the application's availability after deployment.

The main deployment pipeline, defined in *pipeline.yml*, handled the automated build, testing, and deployment sequence.

```yaml
name: Deployment Pipeline
on:
  push:
    branches:
      - main
      - update-workflow-triggers
  pull_request:
    branches: [main]
    types: [opened, synchronize]

jobs:
  build:
    # ... build steps including linting and unit tests ...
  e2e_tests:
    needs: build
    # ... steps to run e2e tests ...
  deployed:
    needs: [build, e2e_tests]
    # ... steps to build client, copy to backend, and deploy backend ...
  tag_release:
    needs: [deployed]
    # ... steps to tag release ...
```

Figure 76: GitHub Actions workflow definition (*pipeline.yml*) outlining the CI/CD process, including build, test (unit and E2E), and conditional deployment jobs to Fly.io.

This pipeline included jobs for building the application, running end-to-end tests, and deploying the backend to Fly.io. Deployment to the main production environment was conditional, occurring only upon successful completion of both the build and E2E testing jobs on the main branch. This gated deployment strategy ensured that only code that passed automated quality checks was pushed to the live service, minimizing the risk of introducing regressions. The pipeline also included steps for dependency installation, code linting, running unit tests, and tagging successful releases. The CI/CD setup, including the secure management of API keys and environment variables through GitHub secrets, was fundamental to maintaining a reliable and efficient deployment workflow.

Looking ahead, future development could focus on scaling the application for broader accessibility. This might involve deploying the application to a dedicated domain and leveraging cloud services like AWS to handle increased user traffic and data storage. Such steps would provide a more robust and professional platform, potentially reaching a larger community of language learners.

6 Results and Evaluation

6.1 Success Indicators and Key Outcomes

Success for this language learning application is measured by how well it helps users to learn languages' vocabulary, improves memory through spaced repetition, and keeps users motivated with gamification and personalization. These goals were outlined in Section 5.1.

To evaluate success, a 15-questions user survey was conducted from March 30, 2025 to April 11, 2025. It collects 3 responses, with participants rating various aspects of the app on a scale of 1 to 10. The questions focused on satisfaction, usability, learning effectiveness, and engagement.

Below is the list of all survey questions along with their rationale and how they help measure success:

Q#	Question	Purpose	Linked Success Indicator
Q1	How satisfied are you with the overall experience of using the app?	Measures general satisfaction	Overall success / UX quality
Q2	How easy was it to navigate and understand the app interface?	Tests usability and UI clarity	Usability improvements (Section 5.2)
Q3	How effective was the app in helping you remember new vocabulary?	Assesses learning impact	Vocabulary retention
Q4	To what extent did the spaced repetition system help reinforce your memory?	Evaluates spaced repetition usefulness	Retention strategy (5.2.2)
Q5	How helpful were the AI-generated translations and explanations?	Tests AI-generated flashcard value	AI-powered learning (5.2.1)
Q6	How motivating did you find the app's XP, streaks, or badge system?	Measures motivation from rewards	Gamification (5.2.3)
Q7	To what extent did features like Matching Game or Flashcard Flipping keep you engaged?	Checks engagement through interactive tools	Gamification & engagement
Q8	How well did the app adapt to your learning needs?	Assess personalization success	Personalization (5.2.4)
Q9	How useful was the Testing page (fill-in-the-blank, synonym matching) in improving your vocabulary?	Tests test-based learning tools	Personalization & effectiveness

Q10	How reliable was the app in terms of performance and speed?	Evaluates backend performance	Technical quality (5.3.3)
Q11	How would you rate the accuracy and clarity of the flashcards?	Further validates AI flashcard quality	Ai translation clarity
Q12	How likely are you to continue using the app?	Measures user retention potential	Long-term engagement
Q13	How frequently did you engage with the app during your test period?	Tests real usage behavior	Actual engagement (vs. Intent)
Q14	How confident do you feel in your vocabulary improvement since using the app?	Measures perceived learning progress	Impact on vocabulary learning
Q15	How likely are you to recommend this app to others learning a language?	Gauges advocacy and trust	Perceived value & user satisfaction

Success indicators of those questions include:

- Vocabulary retention (Q3, Q4)
- Engagement and motivation (Q6, Q7)
- Accuracy of AI-generated flashcards (Q5, Q11)
- Continued usage intent (Q12)

Below are the results based on survey data, highlighting which parts of the app performed well and which areas need improvement.

6.1.1 AI-Powered Flashcards

Indicators: Usefulness of AI translations (Q5), accuracy and clarity of flashcards (Q11)

Linked Figures: See *Appendix 1, Figure A5 and Figure A11*

Outcomes:

- Q5 (Usefulness of AI Translations) scored an average of 7.8/10, and Q11 (Accuracy and Clarity of Flashcards) scored 7.6/10
- As shown in Figure A5 and Figure A11, over 75 % of users rated Q5 above 6, indicating that most found AI-generated translations and explanations helpful.
- However, some users provided lower ratings (5 or below), highlighting occasional vagueness, especially regarding sentence context. These lower score aligns with earlier findings in Section 5.2.1, where the AI struggled with sentence depth and contextual accuracy.
- On the positive side, several users awarded top scores (10/10), emphasizing that when the AI functioned well, it delivered accurate and contextually rich flashcards – particularly in relation to Finnish grammar.

6.1.2 Spaced Repetition System

Indicators: Improved vocabulary memory (Q3), reinforcement through spaced repetition (Q4)

Linked Figures: See *Appendix 1, Figure A3 and Figure A4*

Outcomes:

- Q3 (*How effective was the app in helping you remember new vocabulary?*) averaged 7.9/10, and Q4 (*To what extent did the spaced repetition system help reinforce your memory?*) averaged 7.8/10
- As shown in Figure A3 and A4, most users (over 80 %) rated these questions above 6, indicating that the spaced repetition system effectively supported vocabulary learning.
- Some lower scores (6 or below) were given, possibly due to repeated cards or limited variety – consistent with earlier feedback in Section 5.2.2 regarding over-repetition and scheduling balance.
- Overall, the system performed well and was recognized as a key strength for long-term vocabulary retention.

6.1.3 Gamification

Indicators: Motivation from XP, streaks, badges (Q6), engagement with interactive features (Q7)

Linked Figures: See *Appendix 1, Figure A5* and *Figure A11*

Outcomes:

- Q6 (*How motivating did you find the app's XP, streaks, or badge system?*) scored 7.2/10, while Q7 (*To what extent did features like Matching Game or Flashcard Flipping keep you engaged?*) scored 7.1/10
- As shown in Figure A5 and Figure A11, around 75 % of users gave scores above 6, showing moderate success
- However, several lower scores (5 or below) suggest that gamification was not equally appealing to everyone. This may be linked to earlier issues like the infinite XP bug (see Section 5.2.3) or the fact that some users prefer simpler study methods.

- Still, features like the Matching Game and Flipping Page were well-liked by a smaller group who gave top marks

6.1.4 Personalization

Indicators: Adaption to learning needs (Q8), usefulness of the Testing Page (Q9)

Linked Figures: See *Appendix 1, Figure A8 and Figure A9*

Outcomes:

- Q8 (*How well did the app adapt to your learning needs?*) averaged 7.2/10, and Q9 (*How useful was the Testing page (fill-in-the-blank, synonym matching) in improving your vocabulary?*) averaged 6.8/10 - making personalization the lowest-rated section.
- While many users appreciated the tailoring and test content, or lack of clear benefits.
- These findings match development challenges discussed in Section 5.2.4, where some features required extra effort to use effectively or did not always deliver expected results.

6.1.5 Overall Results

Linked Figures: See *Appendix 1, Figures A1, A2, A10, A12 – A15*

Outcomes:

Overall satisfaction with the app (Q1) averaged 8.0/10, with over 80 % of users rating it above 6 (*Figure A1)*. This suggests that users had a positive experience with the app.

Navigation and usability (Q2) scored even higher at 8.2/10, reflecting improvements made to the layout and responsiveness discussed in Section 5.2 (*Figure A2*).

App performance and speed (Q10) received an average of 7.6/10 (*Figure A10*). While most users were satisfied, a few lower scores hinted at issues with larger decks or slower responses from the API.

The likelihood of continued use (Q12) and recommendation to others (Q15) both averaged 7.8/10, suggesting strong user retention and trust in the app (*Figures A12 and A15)*

Finally, frequency of use (Q13) and confidence in vocabulary improvement (Q14) scored 7.9/10 and 7.8/10, respectively (*Figures A13 and A14)*. These results indicate that users were not only engaged with the app regularly, but also felt it contributed meaningfully to their vocabulary development.

6.2 User Engagement and Learning Impact

The user survey results provide valuable insight into how the app supported learning and kept users engaged over time.

Most users reported positive experiences with key learning features. As shown in Appendix 1, Figures A3 and A4, both vocabulary effectiveness (Q3: 7.9/10) and the spaced repetition system (Q4: 7.8/10) received high scores. These results confirm that the repetition scheduling system played a key role in helping users remember new words, aligning well with the Ebbinghaus forgetting curve model implemented in Section 5.2.2.

AI-generated flashcards also contributed to learning success. Many users appreciated how the appreciated how the app created quick, contextualized cards. As highlighted in Section 6.1, over 75 % of users rated AI-generated translations and explanations above 6 (Q5). This indicates that AI features reduced friction in the learning progress and allowed users to focus more on content than manual translation.

One user noted:

> *"I liked that the app didn't just give me a translation, but also showed small grammar details – like which case was used or why a word changed. It saved me time and helped me actually understand how Finnish works."*
>
> *Anonymous user feedback*

In addition, an external reviewer on GitHub noted the value of this approach, stating:

> *"Utilizing AI-APIs to parse a sentence into its words and recognizing their grammatic forms, it is an interesting and useful learning tool."*
>
> *GitHub review by @ShootingStar91 (see Appendix 2)*

When it comes to engagement, results were more mixed. Gamification elements like XP, streaks, and the Matching Game helped some users stay motivated (Figures A6 and A7), but weren't equally effective for all. While Q6 and Q7 had averages above 7, around one-third of users rated them 6 or lower. This suggests that game-like mechanics added fun and motivation for some, but did not fully capture all users' attention – especially those preferring more traditional learning styles.

As one user shared:

> *"The XP and badges were fun at first, but after a while I just focused on the cards. The learning part was what kept me coming back."*
>
> *Anonymous user feedback*

Interestingly, users still returned to the app regularly even if they weren't highly motivated by gamification. Frequency of use (Q13: 7.9/10) and intent to continue (Q12: 7.8/10) were strong, showing that users valued the core learning experience. This suggests that learning value had a stronger pull than gamification alone.

Personalization was the weakest area in terms of impact. Q8 and Q9 received the lowest averages in the survey, which may explain why a few users didn't feel as engaged. As described in Section

5.2.4, the personalization logic was still under development, and the Testing Page had limited interactivity.

This was reflected in one user's comment:

> *"I wish the app gave me more tailored feedback. Sometimes it felt random what came up next, instead of focusing on what I struggled with"*
>
> *Anonymous user feedback*

In summary, user engagement was mostly driven by effective learning tools, especially spaced repetition and AI flashcards. Gamification had mixed results, and personalization requires more refinement to increase its impact. Overall, users were willing to return to the app regularly and believed it contributed to their language progress.

7 Discussion and Conclusion

Chapter 7 synthesizes the project's journey, offering a comprehensive discussion of its outcomes and concluding perspectives on the development of the multilingual language learning application. The chapter traces the path from initial concept to final implementation, examining the valuable insights gained and the notable obstacles overcome. Experiences and results from preceding chapters are woven together here, bringing into focus the primary lessons derived from the development process and the significant challenges that ultimately shaped this endeavor.

7.1 Key Learnings and Challenges

The creation of the multilingual language learning application was a product-based endeavor, its genesis rooted in addressing distinct challenges identified in the Finnish language learning experience—specifically, the deficiency of contextual vocabulary tools. This section delves into the pivotal learnings and significant challenges that emerged during the intensive coding process, underscoring the inherent technical complexities of constructing and scaling a full-stack application. Insights drawn from user feedback (Appendix 1), an external GitHub review (Appendix 2), and the documented development journey (README.md) illuminate the trajectory of skill development and the practical hurdles surmounted during feature integration.

7.1.1 Key Learnings

The development process yielded several pivotal insights into effective software engineering and project management. A significant realization was the profound impact of modular design on the project's trajectory; the creation of over 30 custom hooks and utility functions proved to be a cornerstone for enhancing code reusability and overall maintainability. Leveraging React's Context API in conjunction with TypeScript, for example, as noted in the external review (Appendix 2), became instrumental in streamlining state management and bolstering type safety across critical components like the flashcard and quiz systems. This commitment to a modular architecture subsequently paid dividends, allowing for the seamless integration of new functionalities, such as the Matching Game and Tailor Decks, without destabilizing existing elements. Such experiences offered a clear reinforcement of the foresight required in planning for extensibility within complex full-stack projects.

Furthermore, the adoption of structured task management methodologies, specifically GitHub's Kanban board and feature branches, brought crucial discipline to the development workflow. This systematic approach enabled the deconstruction of otherwise dauntingly complex features, including the spaced repetition algorithm and the AI-powered flashcard system, into more granular and manageable tasks. The sensible division of the codebase into distinct directories and components, an aspect praised in the GitHub review, indeed fostered an environment conducive to rapid iteration. This iterative cycle, deeply inspired by Agile principles, facilitated continuous refinement driven by user feedback, exemplified by UI clarity improvements made following a reviewer's suggestion to better group on-screen elements.

The establishment of a robust CI/CD pipeline using GitHub Actions, which incorporated over 100 automated tests, was a game-changer for upholding code quality and ensuring the reliability of deployments to Fly.io. Regular daily health checks on the deployed service, coupled with secure management of environment variables for sensitive API keys like OpenAI's, played a vital role in minimizing production errors. This entire experience vividly underscored the indispensable value of automated testing and deployment strategies in preserving application stability, a factor that

became increasingly critical as sophisticated features like gamification and personalization were layered into the system.

A constant theme throughout the coding process was the intricate balance required between managing technical complexity and remaining responsive to user needs. This was particularly evident when developing features such as the Learning Page's customizable quizzes and the Tailor Decks functionality. As highlighted in the project's GitHub README.md (Appendix 3), the prioritization of user-driven features, including the ability to import cards and easily modify decks, stemmed directly from the developer's firsthand experiences and struggles as a language learner. The positive user survey results concerning usability (Section 6.1.5, Q2: 8.2/10), which confirmed the benefits of a responsive and intuitive UI optimized for various devices, served as a welcome validation of this user-centric focus in coding decisions. [The paragraph about the survey limitations and lack of formal statistical testing should be retained as is, as it's a factual statement about methodology.]

Finally, the engagement with external perspectives, such as the GitHub review by user *ShootingStar91* (Appendix 2), proved to be an invaluable source of actionable insights. The recommendation to incorporate *express-async-errors* for streamlined backend error handling, for instance, led to a tangible improvement in code cleanliness and maintainability by reducing repetitive try-catch blocks within Express.JS controllers. This collaborative feedback loop highlighted the often underestimated importance of an outside eye in identifying potential coding inefficiencies and elevating the overall technical implementation.

7.1.2 Challenges

The development journey was characterized by several noteworthy challenges that demanded adaptability and perseverance. The ambition to incorporate engaging features like gamification and sophisticated personalization, for example, inherently introduced layers of coding complexity, sometimes in unexpected ways. Resolving a subtle bug in streak tracking that occurred around midnight necessitated a careful re-evaluation and a shift to calendar-day comparisons (Section 5.2.3), while an infinite XP farming exploit required implementing a *hasAwardedXpToday* flag in *localStorage*. Such instances consistently underscored the critical importance of meticulous state management and rigorous edge-case testing as the application's capabilities expanded.

Navigating frontend intricacies also presented its share of hurdles. Initial efforts to create smooth flashcard flipping animations using CSS transitions proved unexpectedly glitchy, a source of some frustration (as detailed in Section 5.2.3). The solution involved adopting the react-card-flip library, a process that itself presented a learning curve in effectively integrating and customizing external libraries. Furthermore, feedback from the GitHub review (Appendix 2) regarding UI clarity prompted several iterative redesigns with TailwindCSS, serving as potent reminders of the delicate balance required between aesthetic design and functional performance.

Architecting the MongoDB schema robustly enough to support dynamic functionalities, such as the three-state flashcard system and user achievement tracking, posed considerable design challenges. The meticulous process of creating backend diagrams to visualize table connections, while critical, was notably time-consuming. A particularly acute lesson in operational discipline occurred due to an oversight leading to shared MongoDB cluster usage between development and production environments (Section 5.2.3), causing test data leakage and starkly emphasizing the non-negotiable need for strict environment separation and robust database management protocols.

The integration of OpenAI's API, pivotal for the application's core capabilities, also introduced its own unique set of coding challenges. The API's occasional lack of precision in contextual translations, an observation echoed in user feedback (Section 6.1.1, Q5: 7.8/10), necessitated the

development of supplementary web-scraping logic. This improvisation increased backend complexity, as Node.JS scripts were adapted to manage asynchronous API calls and intricate data parsing, thoroughly testing the developer's capacity to handle external dependencies effectively.

Finally, the entire endeavor required surmounting a significant initial learning curve and periods of uncertainty, particularly during an early phase where the project was temporarily abandoned (as described in the README.md). While notebook-based planning for UI and feature evaluation helped, the translation of abstract concepts—such as the spaced repetition algorithm or adaptive testing paradigms—into functional code demanded rapid assimilation of new tools like Vitest and Cypress. This steep learning curve represented a persistent challenge, ultimately mitigated through a strategy of breaking down problems into smaller, more manageable tasks and relying on continuous iterative testing, a problem-solving approach detailed in the README.md.

7.2 Future Development and Research Opportunities

The development of this multilingual language learning application opens several exciting possibilities for future development and research.

7.2.1 Application Enhancements:

The gamification features could be further enhanced by completing the Regional Ranking feature and integrating additional interactive games such as Wordle, thereby providing more diverse and engaging learning experiences.

Based on user feedback, improvements could be made to the spaced repetition system to address areas where it received lower survey scores, potentially leading to increased user satisfaction and learning efficacy.

Future development could focus on scaling the application to accommodate a larger user base and exploring monetization strategies to ensure its sustainability and continued growth.

7.2.2 Research Opportunities:

Further research could involve a longitudinal study to rigorously investigate the long-term impact of AI-driven personalization on vocabulary retention and overall language proficiency.

Research could delve into a comparative analysis of different gamification strategies within language learning applications, aiming to identify optimal approaches for maximizing user engagement and learning outcomes.

Additional research could evaluate the scalability and performance of the application under high user load conditions, providing valuable insights for optimizing deployment and ensuring robust performance in real-world scenarios.

Investigating the potential of sharing the project within the code community as an open-source project or exploring various monetization models could provide valuable insights into sustainable development and dissemination strategies.

7.2.3 Scalability and Deployment:

To effectively handle a growing user base, the application could be migrated to a cloud-based platform like AWS or Google Cloud, leveraging their inherent scalability and reliability.

Expanding the application's API could facilitate integration with other educational platforms and language learning resources, thereby extending its reach and impact.

These future directions offer significant potential to further refine the application, contribute to the field of language learning technologies, and explore sustainable development and dissemination strategies.

7.3 Personal Learning Reflection

The development of the multilingual language learning application has proven to be a deeply enriching and transformative experience.

The project facilitated the acquisition of proficiency in full-stack development, specifically with React.JS, Node.JS, and MongoDB. It also enabled hands-on experience in integrating AI technologies through OpenAI's API and implementing CI/CD pipelines using GitHub Actions.

Throughout the development process, numerous technical challenges were encountered, including inconsistencies in API responses and UI discrepancies. Overcoming these obstacles necessitated the development of robust debugging and problem-solving skills. The subsequent refactoring of the codebase to enhance performance and maintainability further honed analytical and critical thinking abilities.

Managing the project from its initial conception to its final implementation provided invaluable insights into effective project management practices. This included prioritizing tasks, managing time efficiently, and applying Agile methodologies. The iterative nature of the development process and the need to adapt to evolving requirements underscored the importance of flexibility and clear communication.

This thesis has not only expanded technical capabilities but also ignited a stronger passion for creating innovative solutions to real-world challenges. The experience has cultivated a greater sense of confidence as a developer and researcher, and there is a strong desire to leverage these newly acquired skills in future endeavors.

Sources

Anki FAQ. What spaced repetition algorithm does Anki use?. URL: https://faqs.ankiweb.net/what-spaced-repetition-algorithm.html. Accessed: 4th March 2025

Aguion, M. A. R., Baraña, J. A. B., De La Cruz, A., Ilustre, R. G., & Valderrama, C. 2022. Language Acquisition: The Role of Grammar Acquisition and Instruction in Second Language Teaching and Learning. URL: https://www.researchgate.net/publication/356849520_Language_Acquisition_The_Role_of_Grammar_Acquisition_and_Instruction_in_Second_Language_Teaching_and_Learning. Accessed: 17 February 2025

Alhusaiyan, E. 2025. A systematic review of current trends in artificial intelligence in foreign language learning. URL: https://www.emerald.com/insight/content/doi/10.1108/sjls-07-2024-0039/full/pdf?title=a-systematic-review-of-current-trends-in-artificial-intelligence-in-foreign-language-learning. Accessed: 17 February 2025

Altexsoft Editorial Team. 2024. What is API: Meaning, Types, Examples. URL: https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/. Accessed: 25 February 2025

Buljan, M. 2021. Gamification For Learning: Strategies And Examples. URL: https://elearningindustry.com/gamification-for-learning-strategies-and-examples. Accessed: 18 February 2025

Butler, A.C., Henry, R.L. 2011. The critical role of retrieval practice in long-term retention. URL: https://www.sciencedirect.com/science/article/pii/S1364661310002081. Accessed: 4th March 2025

Blanco, C. 2024. Duolingo 101: How to learn a language on Duolingo. URL: https://blog.duolingo.com/duolingo-101-how-to-learn-a-language-on-duolingo/. Accessed: 4th March 2025

Brown, S. 2021. Machine learning, explained. URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. Accessed: 18 February 2025

Cepeda, J.N., Pashler, H., Vul, Edward., Rohrer, D., & Wixted, J.T. 2006. Distributed Practice in Verbal Recall Tasks: A Review and Quantitative Synthesis. URL: https://www.yorku.ca/ncepeda/publications/CPVWR2006.pdf. Accessed: 4th March 2025

Codeacademy Team s.a. MVC: Model, View, Controller. URL: https://www.codecademy.com/article/mvc. Accessed: 27 February 2025

Csikszentmihalyi, M. (1990). Flow: The Psychology of Optimal Experience. Harper & Row.

New York, NY.

Deci, E. L., Koestner, R., & Ryan, R. M. (1999). A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. Psychological Bulletin, 125(6), 627-688. URL: https://doi.org/10.1037/0033-2909.125.6.627. Accessed: 10 March 2025

Ebbinghaus, H. 1885. Memory: A Contribution to Experimental Psychology. URL: ia800802.us.archive.org/26/items/memorycontributi00ebbiuoft/memorycontributi00ebbiuoft.pdf. Accessed: 4 March 2025

Express. 2024. Express – Node.js web application framework. URL: https://expressjs.com/. Accessed: 25 February 2025

Fiorella, L. & Mayer, R. 2020. What works in generative learning: Eight strategies that promote understanding. Educational Psychology Review, 32(4), 569–586. URL: https://www.researchgate.net/publication/264233729_Learning_as_a_Generative_Activity_Eight_Learning_Strategies_that_Promote_Understanding. Accessed: 16 April 2025

GeeksforGeeks. 2025. JSON Web Token (JWT). URL: https://www.geeksforgeeks.org/json-web-token-jwt/. Accessed: 25 February 2025

Google Cloud. What Is Artificial Intelligence (AI)?. URL: https://cloud.google.com/learn/what-is-artificial-intelligence?hl=en. Accessed: 17 February 2025

Hall, S. 2024. Getting Started With MongoDB & Mongoose. URL: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/. Accessed: 27 February 2025

Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does gamification work? – A meta-analysis of empirical studies on gamification. In *Proceedings of the 47th Hawaii International Conference on System Sciences* (pp. 3025–3034). Washington, DC: IEEE Computer Society. URL: https://ieeexplore.ieee.org/document/6758978. Accessed: 7 March 2025.

Holdsworth, J., Stryker, C. 2024. What is NLP (Natural Language Processing)? | IBM. URL: https://www.ibm.com/think/topics/natural-language-processing. Accessed: 18 February 2025

Jojo. 2025. Segment Leaderboard Guidelines. URL: https://support.strava.com/hc/en-us/articles/216919507-Segment-Leaderboard-Guidelines. Accessed: 10 March 2025

Kang, S. H. K., 2016. Spaced Repetition Promotes Efficient and Effective Learning: Policy Implications for Instruction. URL: https://www.researchgate.net/publication/290511665_Spaced_Repetition_Promotes_Efficient_and_Effective_Learning_Policy_Implications_for_Instruction. Accessed: 4th March 2025

Kontra, C., Goldin-Meadow, S., & Beilock, S. L. (2015). *Embodied learning across the life span*. Topics in Cognitive Science, 7(4), 731–739. URL: https://doi.org/10.1111/tops.12157. Accessed: 04 April 2025

Kwantlen Polytechnic University. Spaced Repetition: Remembering What You Learn. URL: https://www.kpu.ca/sites/default/files/Learning%20Centres/Think_SpacedRepetition_LA.pdf. Accessed: 18 February 2025

Memrise. 2024. How does the spaced repetition system work?. URL: https://memrise.zendesk.com/hc/en-us/articles/360015889057-How-does-the-spaced-repetition-system-work. Accessed: 4th March 2025

Monclair State University. Adaptive Learning. URL: https://www.montclair.edu/itds/digital-pedagogy/pedagogical-strategies-and-practices/adaptive-learning/#:~:text=Adaptive%20learning%20is%20a%20technique,to%20provide%20personalized%20learning%20experiences. Accessed: 17 February 2025

Mucci, T. 2024. What is data-driven decision-making?. URL: https://www.ibm.com/think/topics/data-driven-decision-making. Accessed: 18 February 2025

Nguyen, T. 2019. Assessment strategies in Vietnamese education. Journal of Educational Methods, 12(3), 45-60. Accessed: 11 March 2025

Node.js. 2025. Node.js - About Node.js®. URL: https://nodejs.org/en/about#about-nodejs. Accessed: 26 February 2025

OpenAI Platform. 2025. API Reference – OpenAI API. URL: https://platform.openai.com/docs/api-reference/introduction. Accessed: 26 February 2025

Pavlik, P. I., & Anderson, J. R. 2008. Using a model to compute the optimal schedule of practice. URL: https://eric.ed.gov/?id=EJ802557. Accessed: 4th March 2025

React s.a. Quick Start – React. URL: https://react.dev/learn. Accessed: 26 February 2025

Roediger, H. L., & Butler, A. C. 2011. The critical role of retrieval practice in long-term retention. Trends in Cognitive Sciences. URL: https://doi.org/10.1016/j.tics.2010.09.003. Accessed: 04 April 2025

Skitter. 2024. Spaced Repetition System (SRS). URL: https://docs.skritter.com/article/250-spaced-repetition-system#:~:text=A%20Spaced%20Repetition%20System%2C%20commonly,a%20learner%20to%20remember%20it. Accessed: 22 February 2025

Wikipedia. Digital learning. URL: https://en.wikipedia.org/wiki/Digital_learning. Accessed: 18 February 2025

Wikipedia. Don’t repeat yourself. URL: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself. Accessed: 21 April 2025.

Wikipedia. General Data Protection Regulation. URL: https://en.wikipedia.org/wiki/General_Data_Protection_Regulation. Accessed: 22 April 2025

Wilson, M. (2002). *Six views of embodied cognition*. Psychonomic Bulletin & Review. URL: https://doi.org/10.3758/BF03196322. Accessed: 04 April 2025

Wozniak, P. 1990. Optimization in learning. URL: https://www.supermemo.com/en/blog/optimization-of-learning. Accessed: 4th March 2025.

Appendices

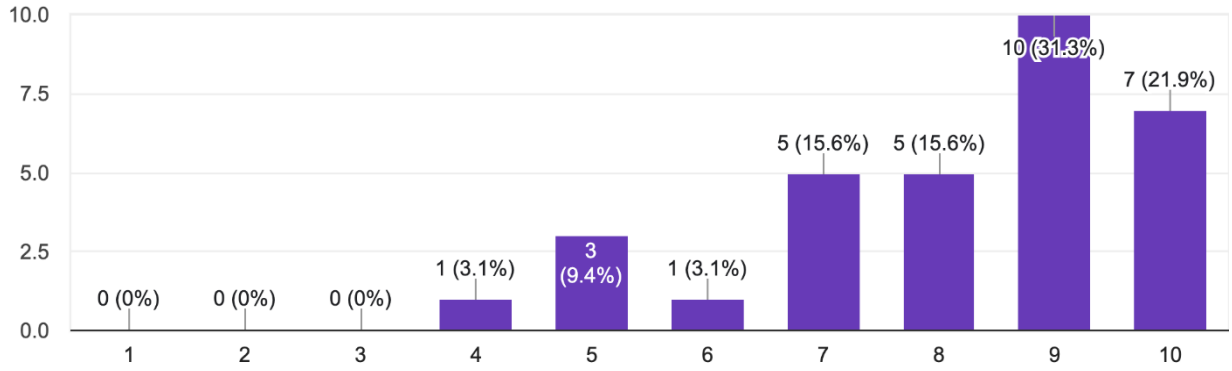
Appendix 1. Survey Results – Visual Summary

This appendix presents the full set of survey charts related to user feedback, as referenced in Chapter 6. Each figure corresponds to a specific survey question (Q1 – Q15) and visualizes trends such as average ratings, response distribution, or engagement levels.

Figure	Description	Question
A1	Overall satisfaction with the app	Q1
A2	Ease of navigation and interface usability	Q2
A3	Effectiveness in remembering vocabulary	Q3
A4	Spaced repetition impact	Q4
A5	Helpfulness of AI-generated translations	Q5
A6	Motivation from XP, streaks, badges	Q6
A7	Engagement from Matching Game and Flashcards	Q7
A8	Adaptation to learning needs	Q8
A9	Usefulness of Testing Page	Q9
A10	App reliability and speed	Q10
A11	Accuracy and clarity of flashcards	Q11
A12	Likelihood of continued use	Q12
A13	Frequency of app usage during test period	Q13
A14	Confidence in vocabulary improvement	Q14
A15	Likelihood of recommending the app	Q15

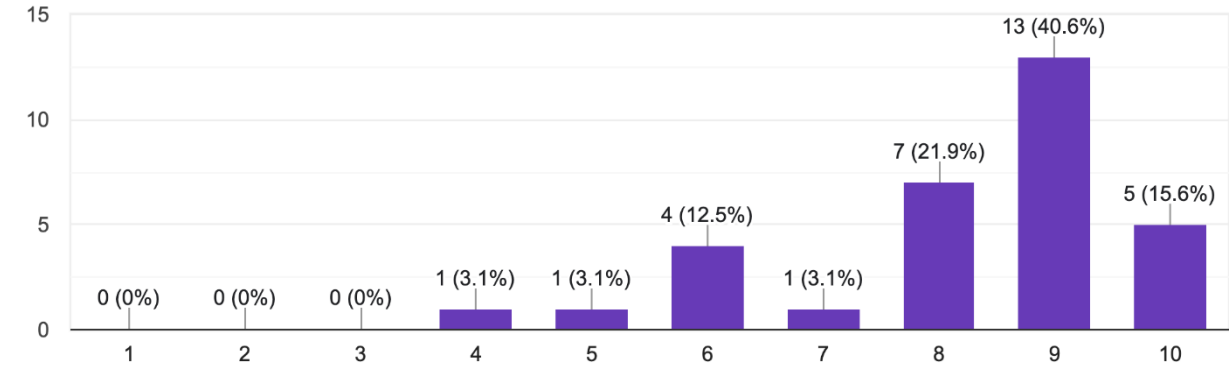
1. How satisfied are you with the overall experience of using the language learning app?

32 responses



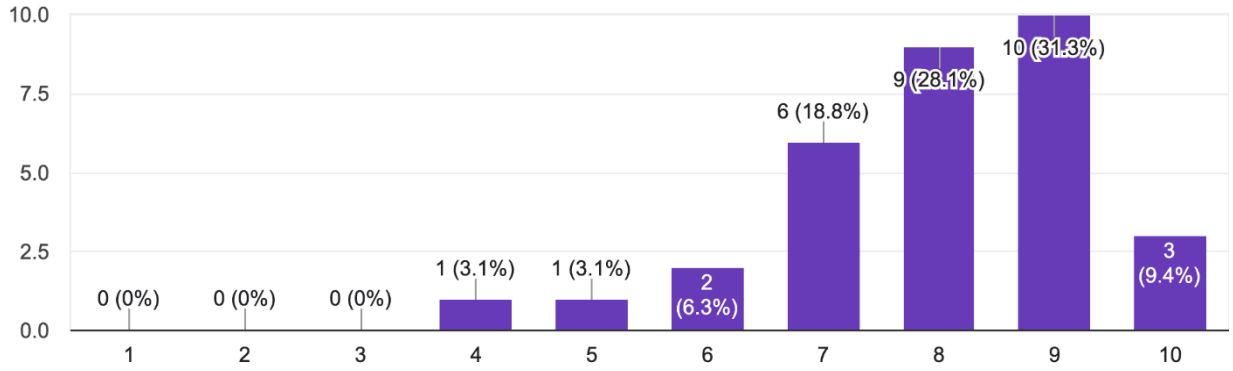
2. How easy was it to navigate and understand the app interface?

32 responses



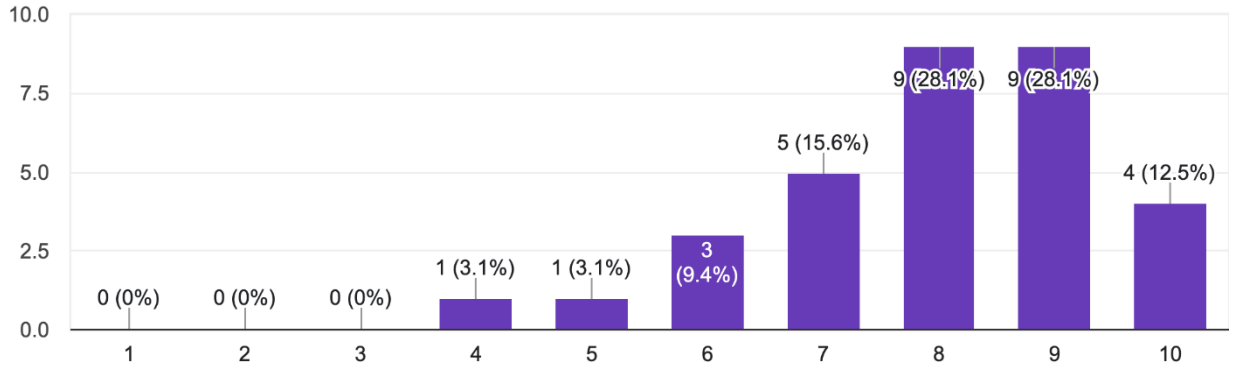
3. How effective was the app in helping you remember new vocabulary over time?

32 responses



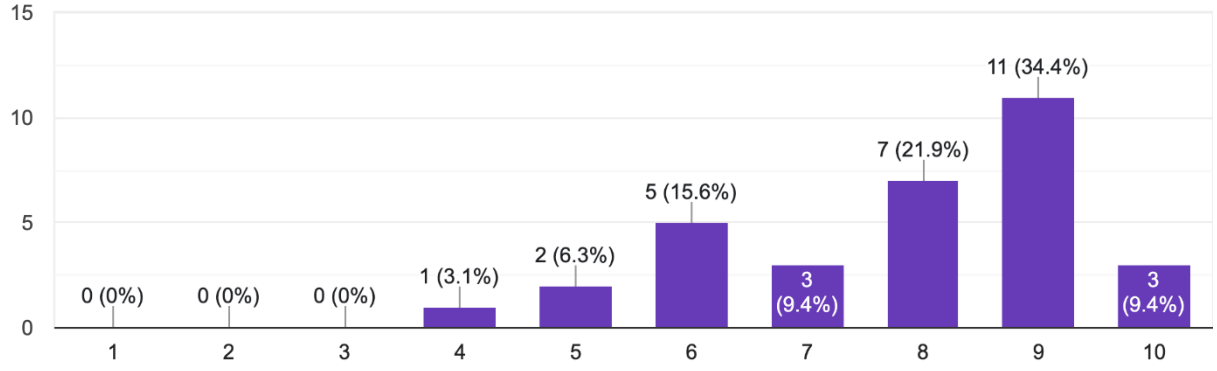
4. To what extent did the spaced repetition system help reinforce your memory?

32 responses



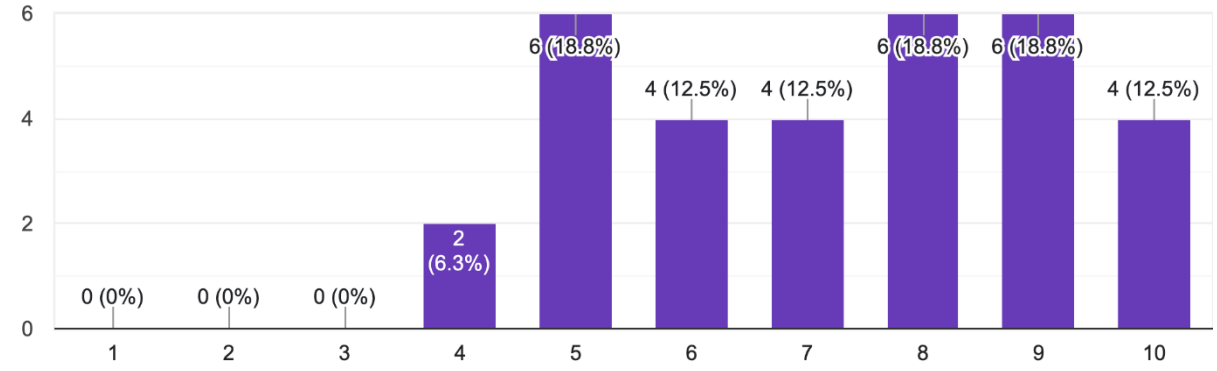
5. How helpful were the AI-generated translations and explanations in understanding sentence meaning?

32 responses



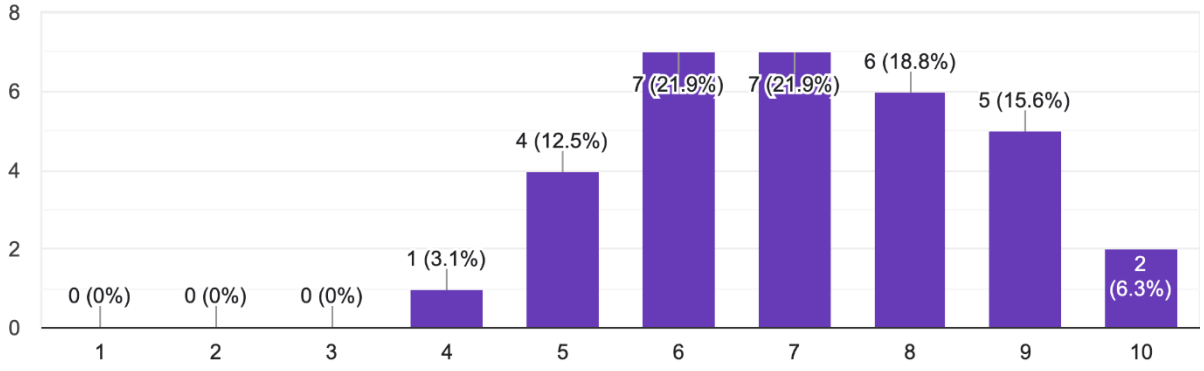
6. How motivating did you find the app's XP, streaks, or badge system?

32 responses



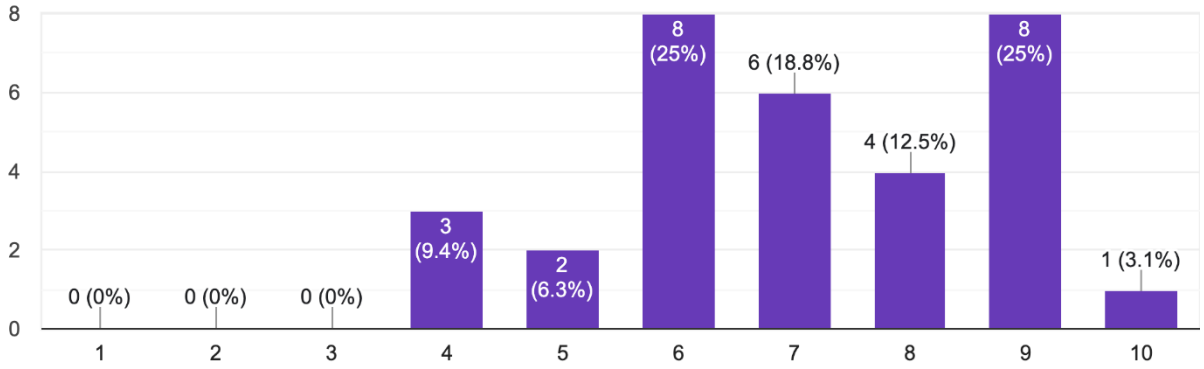
7. To what extent did gamification elements like the matching game or flashcard flipping increase your engagement?

32 responses



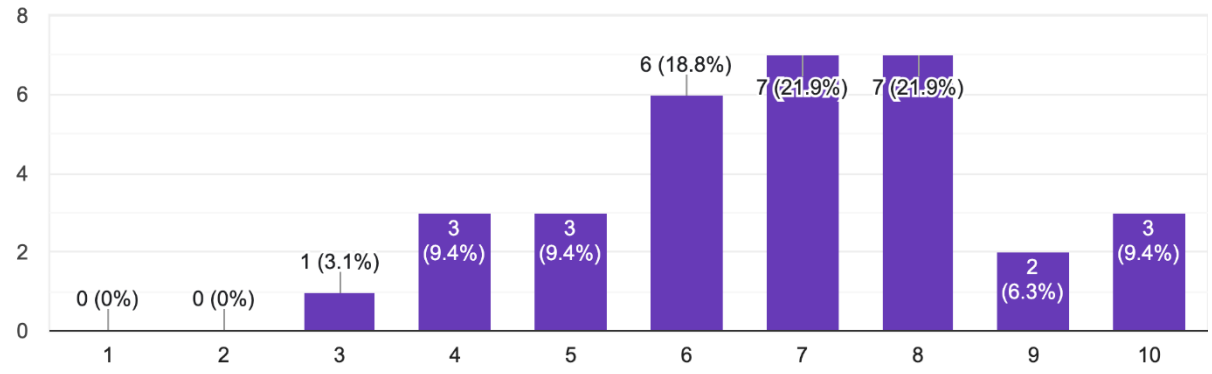
8. How well did the app adapt to your learning needs (e.g., tailoring quizzes, AI-suggested corrections)?

32 responses



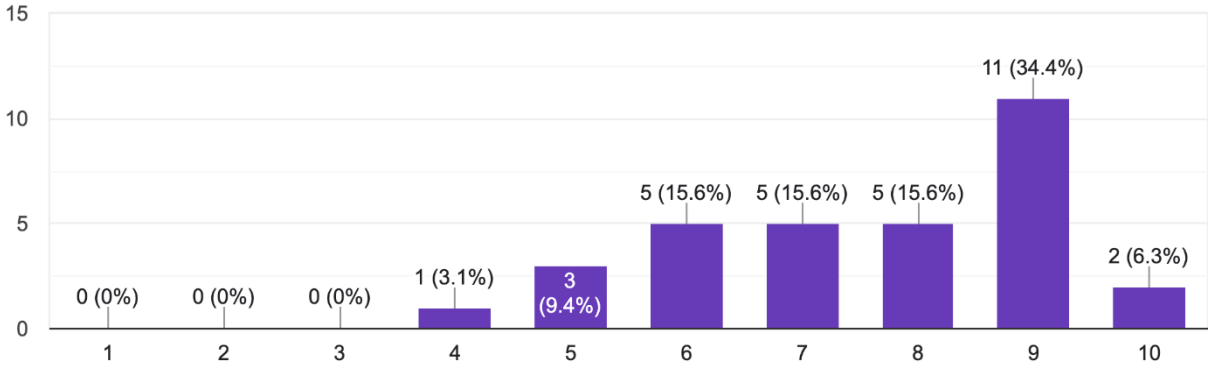
9. How useful was the Testing Page (e.g., fill-in-the-blank or synonym matching) in enhancing your vocabulary?

32 responses



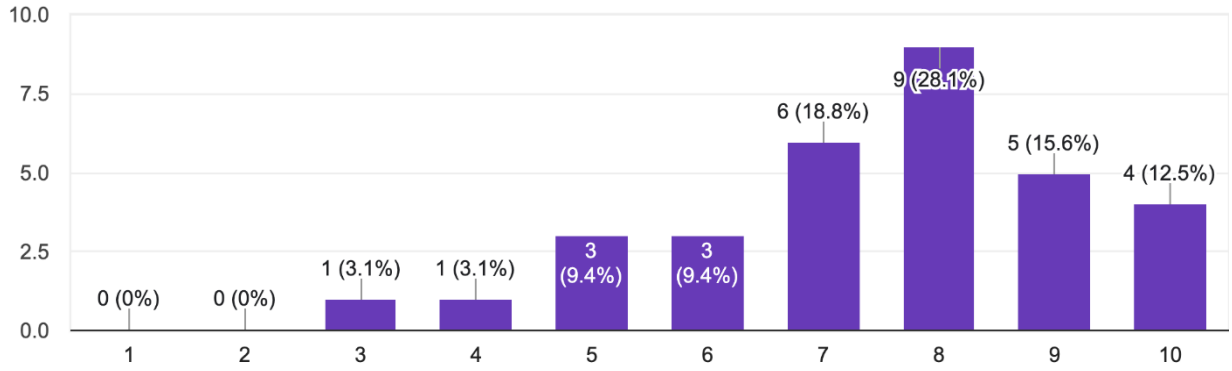
10. How reliable was the app in terms of performance and speed?

32 responses



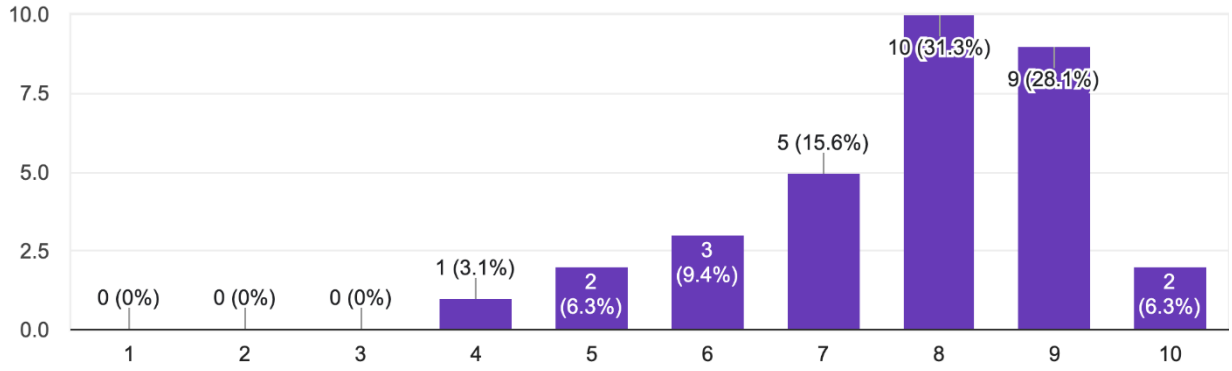
11. How would you rate the accuracy and clarity of AI-generated flashcards?

32 responses



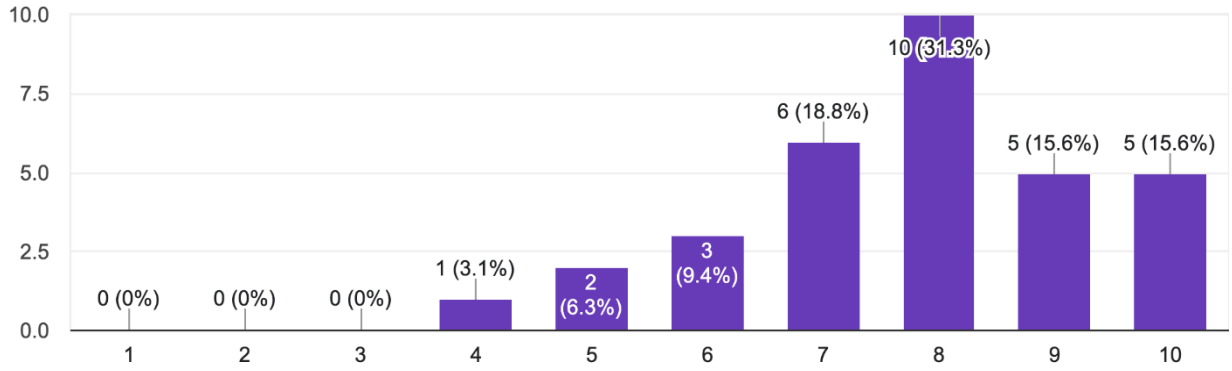
12. How likely are you to continue using the app in the future?

32 responses



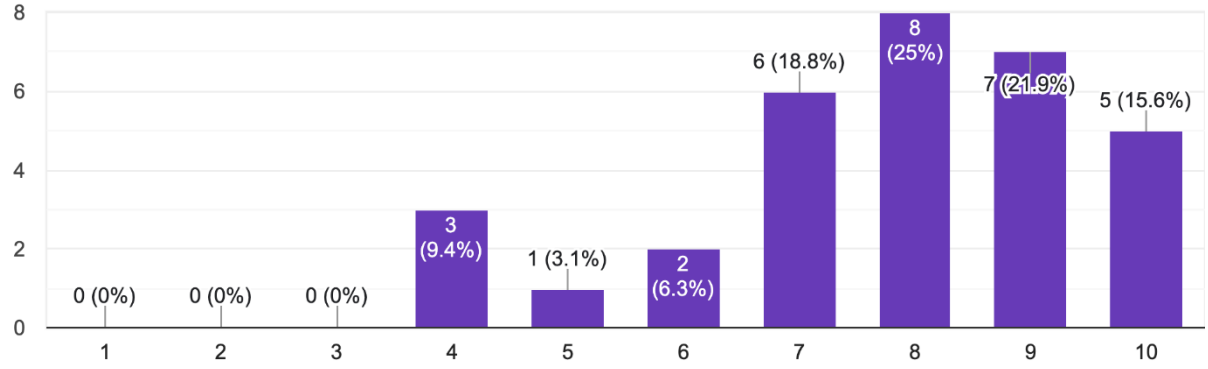
13. How frequently did you engage with the app during your testing period?

32 responses



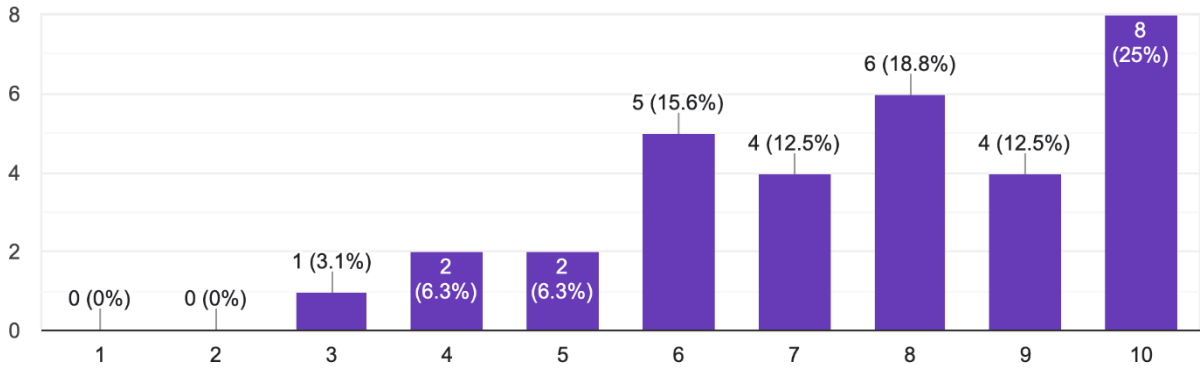
14. How confident do you feel in your vocabulary improvement since using the app?

32 responses



15. How likely are you to recommend this app to others learning a language?

32 responses



Appendix 2. External Project Review – GitHub Feedback by @ShootingStar91

This review was written by GitHub user @ShootingStar91 as part of the Fullstack Project evaluation on 20th July 2024. The review provides feedback on both the user experience and technical implementation of the app. It was based on hands-on testing and a code-level inspection.

Original GitHub Issue: https://github.com/padwhen/language-learning-app/issues/36

“Fullstack-project review

This is a review of your Fullstack practice project. You may apply the suggestions presented here if you so wish, but you do not have to implement any further changes to receive the credits for your project.

User experience

Translating a sentence worked nicely. There is an estimate and a loading icon. I like that the main feature is available as soon as I open the app. The login and registration could be done after doing the translation, without having to move away from the page. This was a smooth user experience that invited me to use the app instead of having to think about whether I should first register, etc.

The UI is stylish and clean. Some improvements could be better dividing areas to group items together, in some places it's a bit confusing with a lot of different texts and items close to each other with no clear grouping. Also, there is not enough margin at the bottom of the page - see the picture as an example for both things (screen is scrolled as far down as possible).

The navigation bar on the top where I see "Deck > Matchgame" etc is nice. However, it could be there on the front page too. At first navigating felt a bit confusing.

There are many features. Flashcards work nicely, though clicking the pen or audio icons still do not seem to work, they just flip the flashcard.

The Match Game is a nice idea. It works well. However, the "Return to Deck Details" and "Options" links do not work.

Code

The code is readable and mostly well divided in sensible directories, files and components

Data fetching is sometimes done within the component, with state-setting and error-handling. This is usually best done with some external library like Tanstack Query to prevent having to repeat try-catch, state-setting and possible auth headers for every query separately.

Points for using TypeScript!

Context API and custom hooks utilized well

Nice usage of TailwindCSS for styling

For the backend, I strongly recommend using npm-package 'express-async-errors'. It makes the server catch the errors thrown by calls to external services and moves them to be handled by the middleware, and you don't have to spam try-catch on every controller, making the code much cleaner and cognitively simpler.

Summary

The project is an impressive application for language learning. Utilizing AI-API's to parse a sentence into it's words and recognizing their grammatic forms, it is an interesting and useful learning tool. There are multiple features. The first time I tried the application, a lot of things weren't working. Now most were fixed but there were still some links that did nothing. I know from experience that it's important to test the app thoroughly also in production. But despite the initial trouble this application is impressive in it's scope and idea. Very good job!”

Appendix 3. GitHub Repository

The source code for the language learning application developed in this thesis is available at:

https://github.com/padwhen/language-learning-app

Appendix 4. Initial Algorithm Planning Notes

“First quiz: Answer 8 terms

Initial score: 1,2,3,5,6,7,8: 1 point. 4: 2 points. Cards with 5 points are not included.

1	2	3	4	5	6	7	8

Score 0-2: High Priority

Score 3-4: Review Again

Score 5: Completed

High Priority → Review twice

Review again → Review once

- Cards with scores 0-2 are always in High Priority, regardless of correctness or time taken.
- Cards with scores 3-4 are in Review Again.
- However, cards with scores 3-4 with slow time will be in High Priority as well

Card 4 is in the ‘Review Again’ section. The others are in the ‘High Priority’ section. Meaning that those will be reviewed twice.

1	1	2	2	3	3	4	5
5	6	6	7	7	8	8	

The score is already being modified in the first quiz.

If card appears two times:

- Answer one correct, answer one incorrect ⇒ Score stay the same
- Answer two correct ⇒ Score ++
- Answer two incorrect ⇒ Score minus 1

If card appear one time:

- Answer incorrect ⇒ Score - 2
- Answer correct ⇒ Score + 1

First quiz will be marked as ‘Learn’.

From the second quiz will be marked as ‘Review’

When all card is reviewed to score 5, it will be marked as ‘Completed’

→ Cards in this quiz will not be included in another ‘Learn’ section. Meaning it will be excluded from ‘All cards’ for future learning.

If the quiz is not reviewed according to the deadline within 24 hours, all scores of each card will be minus by 1.

If all the cards in this quiz reach 0, it will be added back to the ‘All cards’”