# Adaptive Learning Systems for Middle School Math

BSc Thesis

William Peytz
Technical University of Denmark (DTU)

June 22, 2025

**Abstract**

This thesis explores the design, implementation, and evaluation of *EloQuiz*, an adaptive learning platform aimed at middle school mathematics education. The system leverages the Elo rating algorithm, traditionally used in competitive games, to match students with appropriately challenging quiz questions in real time. Questions are either retrieved from a curated database or dynamically generated using OpenAI's language models, ensuring content diversity and adaptability.

The study investigates whether an Elo-based system can accurately assess student proficiency and maintain engagement by tailoring difficulty levels. A mixed-methods approach was used, combining classroom testing with quantitative performance metrics and qualitative feedback from students and teachers. Data from nine student participants across two 5th-grade classes showed that the Elo-based adjustment mechanism could produce differentiated scores across mathematical categories, aligning with perceived topic difficulty.

While technical limitations and small sample size constrained the generalizability of findings, results suggest that EloQuiz holds promise as a lightweight, scalable adaptive learning tool. The system also aligns with ethical guidelines set forth by the EU AI Act, emphasizing transparency and non-intrusiveness in educational settings. Future work should include long-term studies, larger cohorts, and further exploration of the platform's applicability beyond mathematics.

# Contents

# 1 Introduction

## 1.1 Motivation

Effective personalized learning is a critical challenge in education, particularly in subjects like mathematics, where students have varying levels of proficiency. Traditional classroom teaching often relies on a one-size-fits-all approach, which can lead to disengagement among students who find the material either too easy or too difficult. Adaptive learning systems offer a promising solution by dynamically adjusting question difficulty based on student performance. However, ensuring that students are consistently challenged at an appropriate level remains a complex problem.

One potential approach to solving this issue is the use of an Elo-style rating system, originally developed for ranking players in competitive games such as chess (Elo [1978]). In this framework, both students and questions are assigned a rating, which updates dynamically based on student performance. If a student successfully answers a question, the student's rating increases slightly while the question's difficulty decreases slightly, and vice versa. This method allows for a data-driven adaptation of difficulty, ensuring that each student receives problems that are neither too easy nor too difficult (Glickman [1995]).

Despite the widespread success of Elo in competitive ranking systems, its application in adaptive learning environments remains an open question. While some adaptive learning models exist, many rely on pre-defined rules rather than real-time adjustments based on student performance. Thus, it is unclear whether an Elo-based difficulty adjustment system can accurately reflect student proficiency and continuously optimize learning progression.

This study is guided by the following research question:

**"Can an Elo rating system accurately predict a student's proficiency level and provide optimally challenging questions?"**

To answer this question, an adaptive learning platform that dynamically adjusts question difficulty based on an Elo-style rating system was developed. By analyzing student

performance data, it aims to evaluate whether Elo can effectively match students with questions at an optimal difficulty level. The findings from this study could contribute to the development of more efficient and scalable adaptive learning methodologies

## 1.2 Scope and focus

This project explores the development of an adaptive learning system for middle school mathematics, with a specific emphasis on dynamic question difficulty adjustment using an Elo-style rating system. The focus lies in the technical implementation, user-facing functionality, and evaluation of how effectively the system can tailor learning experiences based on student performance.

The platform is designed for Danish school contexts, targeting 5th-grade students. It supports both students and teachers through a web interface and includes backend features for content generation, performance tracking, and data storage.

The scope of the study is limited to:

- Designing and implementing the core functionality of the adaptive system.
- Generating multiple-choice questions in Danish using a large language model (GPT).
- Evaluating the platform through short-term classroom testing with real users.
- Measuring usability, performance, and perceived effectiveness based on defined evaluation criteria.

The project does not aim to provide a complete curriculum or replace traditional instruction. Nor does it attempt to rigorously validate the Elo algorithm in long-term educational settings. Instead, the goal is to demonstrate a working proof-of-concept and gather preliminary insights into its potential value in personalized education.

## 1.3 Approach overview

The approach taken in this project combines system design, implementation, and evaluation to investigate the use of an Elo-style adaptive learning model in a real-world educational context. The methodology follows a structured progression from conceptualization to deployment and testing.

1. **System design:** The first phase focused on defining the platform architecture, including the roles of the student interface, Elo rating engine, OpenAI-powered question generation, and teacher dashboard. Design decisions were made with usability, adaptability, and scalability in mind.

2. **Implementation:** A full-stack web application was developed using Firebase for backend services and a custom frontend interface for students and teachers. The platform integrates real-time data updates, automatic difficulty calibration using Elo scores, and API-based question generation.

3. **User testing:** The system was deployed in a classroom setting with 5th-grade students, where participants interacted with the platform over a single session. Data was collected on engagement, performance, and user feedback.

4. **Evaluation:** Both qualitative and quantitative metrics were analyzed to assess the accuracy of the adaptive system, user experience, and the reliability of the question generation pipeline. Specific evaluation criteria were defined in advance to ensure consistency.

This structured approach ensures that the research question is addressed through both technical development and empirical validation, balancing implementation challenges with insights from real user interaction.

## 1.4 Thesis structure

The remainder of this thesis is organized as follows:

- **Chapter 2: Background**
  Provides an overview of key concepts relevant to the project, including adaptive learning theory, the Elo rating system, large language models, and existing educational platforms.

- **Chapter 3: Methodology**
  Describes the research design, including the research question, hypothesis, and evaluation strategy. This chapter also introduces the system architecture and outlines the user testing process.

- **Chapter 4: Implementation**
  Details the technical realization of the system, including backend architecture, frontend development, data handling, and the integration of the OpenAI API.

- **Chapter 5: Results and evaluation**
  Presents findings from the user test and analyzes system performance based on defined evaluation criteria. Includes both quantitative data and qualitative user feedback.

- **Chapter 6: Discussion**
  Reflects on the results, explores the limitations of the study, and discusses the broader implications for adaptive learning technologies.

- **Chapter 7: Conclusion**
  Summarizes the main contributions of the thesis and suggests directions for future development and research.

This structure is intended to guide the reader logically through the development, analysis, and reflection on the adaptive learning platform.

# 2 Background

## 2.1 Adaptive learning theory

Adaptive learning refers to educational methods and technologies that dynamically tailor content and instruction to the individual needs of learners. This approach stands in contrast to traditional one-size-fits-all instruction, which often fails to account for differences in student background, learning pace, and prior knowledge. By personalizing the learning experience, adaptive systems aim to maintain optimal engagement and improve learning outcomes.

Modern adaptive systems typically adjust the difficulty, sequencing, or presentation of educational materials based on real-time performance data. These systems leverage various techniques, ranging from simple rule-based heuristics to more sophisticated models like Bayesian Knowledge Tracing (BKT), Item Response Theory (IRT), and reinforcement learning algorithms. The goal is to maintain an appropriate level of challenge avoiding both boredom from overly simple tasks and frustration from tasks that are too difficult.

In mathematics education specifically, adaptive learning has shown promise in addressing disparities in skill level and engagement, particularly when students receive feedback that matches their current proficiency. However, designing systems that can both assess and respond to student ability in real time remains a significant challenge. This difficulty underscores the need for robust learner models capable of dynamically updating based on performance, a problem this thesis aims to address through the use of Elo-style rating systems.

## 2.2 Elo rating for education

Originally developed to rank chess players, the Elo rating system provides a simple, dynamic method for estimating the relative skill of individuals based on performance outcomes (Elo [1978]). Each participant is assigned a numerical rating, and this value is updated after each "match" based on whether the participant performed better or worse than expected. The magnitude of the update depends on a tunable parameter known as the K-factor, which controls how responsive the rating system is to new information

(Glickman [1995]).

In educational settings, Elo-style systems can be adapted to model both student ability and question difficulty. When a student answers a question, the system interprets this interaction as a match between two "players": the student and the question. A correct answer results in an increase in the student's rating (and potentially a decrease in the question's difficulty rating), while an incorrect answer has the opposite effect. This formulation creates a data-driven feedback loop that adjusts both student and question ratings over time.

Compared to more complex models such as Item Response Theory (IRT) or Bayesian Knowledge Tracing (BKT), Elo offers a computationally lightweight and intuitive approach that is particularly well-suited for online adaptation in interactive learning environments. It allows real-time updates without requiring large datasets or extensive parameter tuning, making it attractive for systems where rapid feedback is essential.

However, the application of Elo in education is not without limitations. The model assumes that ability and difficulty can be captured as single scalar values, which may oversimplify complex cognitive processes.

Despite these challenges, the Elo framework remains a compelling choice for adaptive systems seeking a balance between interpretability, responsiveness, and scalability.

## 2.3 Elo rating formula

The Elo rating system updates a participant's rating based on the difference between the expected and actual outcome of an interaction. The update formula is:

$$R_{\text{new}} = R_{\text{old}} + K(S - E) \tag{2.1}$$

Where:

- $R_{\text{new}}$ is the updated rating
- $R_{\text{old}}$ is the current rating
- $K$ is the update factor (controls sensitivity)
- $S$ is the observed outcome (1 for correct, 0 for incorrect)
- $E$ is the expected outcome, calculated as:

$$E = \frac{1}{1 + 10^{(R_{\text{opponent}} - R_{\text{old}})/400}} \tag{2.2}$$

**The $K$ parameter:**
The $K$ factor determines how significantly a single interaction affects the rating. Higher

values of $K$ result in ratings that adjust more rapidly, which is suitable for new users or initial assessments, while lower values provide stability and are commonly used after a participant's skill level is well-established. Typical values range between 16 (conservative adjustments) and 32 (more responsive adjustments). For EloQuiz, a value of $K = 32$ was selected to enable quicker adaptation during brief interactions typical in classroom settings.

**Example:**
Suppose a student has a rating of 1300 and attempts a question rated at 1400, with $K = 32$. If the student answers correctly ($S = 1$):

$$E = \frac{1}{1 + 10^{(1400-1300)/400}} = \frac{1}{1 + 10^{0.25}} \approx 0.36$$

$$R_{\text{new}} = 1300 + 32 \cdot (1 - 0.36) = 1300 + 20.48 = 1320.48$$

The student's rating increases by approximately 20.5 points.

Since the student answered correctly, the question "lost." Its updated rating is calculated similarly, but with $S = 0$:

$$E = \frac{1}{1 + 10^{(1300-1400)/400}} = \frac{1}{1 + 10^{-0.25}} \approx 0.64$$

$$R_{\text{new}} = 1400 + 32 \cdot (0 - 0.64) = 1400 - 20.48 = 1379.52$$

The question's rating decreases by approximately 20.5 points, indicating it was easier than its initial Elo score suggested.

## 2.4 Large language models and GPT

Large language models (LLMs) are a class of neural networks trained to predict the next token in a sequence of text. By processing vast amount of written language, these models learn statistical patterns and structures in natural language, enabling them to generate coherent and contextually relevant responses to user prompts. Recent advancements in LLMs have been driven by transformer-based architectures, most notably the Generative Pre-trained Transformer (GPT) series developed by OpenAI.

GPT models are trained in two phases: a pretraining phase where the model learns general language representations from a broad dataset, and a fine-tuning or instruction-

following phase that aligns the model with specific user tasks. The resulting models, such as GPT-3.5 and GPT-4, are capable of a wide range of applications, including summarization, translation, question-answering, and content generation. Importantly, these models can generate responses with a high degree of fluency and contextual awareness, even in tasks for which they were not explicitly trained.

In educational contexts, LLMs have shown potential for generating learning materials, providing explanations, and facilitating personalized tutoring experiences. Their ability to produce custom question content on demand makes them particularly useful for adaptive learning systems, where content must be varied and appropriately challenging. However, LLMs also pose challenges, including the risk of generating incorrect or misleading information ("hallucinations"), inconsistencies in formatting, and the lack of explicit control over content difficulty Zhao et al. [2023]. As such, their integration into learning platforms must be carefully managed to ensure the reliability and pedagogical value of the generated content.

This thesis leverages OpenAI's GPT models via API to generate mathematical quiz questions tailored to middle school students. The use of an LLM enables scalable content generation, while the Elo-based adaptation system is used to assess and align the difficulty of these questions with student proficiency.

## 2.5 Using APIs for question generation

Large language models such as GPT-4 are typically accessed via web-based APIs, allowing developers to send prompts and receive generated responses in real time. This architecture enables integration of LLMs into third-party applications without requiring local deployment or significant computational resources. For adaptive educational platforms, API-based content generation provides a scalable way to produce varied and personalized learning material on demand.

In this project, the OpenAI API is used to generate math questions suitable for middle school students. Prompts are crafted dynamically based on topic tags and student performance, with the model instructed to return output in a structured JSON format, which enable seamless integration with the rest of the platform particularly the Elo-based difficulty adjustment system and the teacher-facing interface.

However, generating questions through a language model API introduces several challenges. First, the probabilistic nature of LLMs can lead to variability in output, including formatting errors or factual inaccuracies commonly referred to as "hallucinations" (Zhao et al. [2023]). Second, API usage comes with practical constraints such as token limits, rate throttling, and cost per request. To mitigate these issues, this system includes a manual validation interface for teachers to remove low-quality questions, and may later incorporate automated checks for format compliance and mathematical correctness.

Despite these limitations, the use of LLM APIs offers a powerful and flexible approach to question generation. By combining prompt design, structured output, and validation mechanisms, the system achieves a balance between scalability and content reliability.

## 2.6 Existing educational platforms

A number of widely used educational platforms employ adaptive learning techniques to personalize instruction and assessment. Systems such as Khan Academy and Duolingo, dynamically adjust content based on learner performance, leveraging concepts such as spaced repetition, skill mastery tracking, and personalized progress paths. These platforms aim to maintain learner engagement by continuously aligning task difficulty with a student's estimated proficiency.

Khan Academy, for example, uses a mastery-based system where students progress through skill units by demonstrating consistent accuracy. Duolingo incorporates reinforcement principles by using experience points, streaks, and adaptive lesson sequencing based on previous performance. While these platforms employ forms of adaptation, the underlying models are often proprietary and rely on rule-based or hand-tuned progression mechanisms rather than real-time rating systems like Elo.

Moreover, most existing platforms rely on pre-authored question banks, which limits their flexibility in generating novel or context-specific content. While this ensures content quality and alignment with curriculum standards, it also places a ceiling on scalability and personalization. In contrast, this project explores the use of large language models (LLMs) to generate question content on demand, thereby expanding the range of available material and allowing for a more fluid, scalable interaction between student ability and content difficulty.

By combining dynamic content generation with real-time difficulty adaptation through Elo-style rating, the proposed system offers a lightweight, extensible alternative to more rigid platforms. While existing systems provide proven frameworks for large-scale education delivery, the approach developed here emphasizes modularity, scalability, and experimentation offering a sandbox for exploring new ways to personalize learning.

# 3 Methodology

This chapter outlines the methods used to design, develop, and evaluate *EloQuiz*, a web-based adaptive learning platform aimed at middle school math education. The primary objective was to investigate whether an Elo-style rating system could accurately assess student proficiency and deliver questions of appropriate difficulty in real time. To address this, the methodology combines system design, technical implementation, and real-world user testing in classroom environments.

The approach integrates both quantitative and qualitative evaluation strategies. Quantitative data was gathered from system interactions, including Elo progression and answer accuracy, while qualitative insights were collected through student and teacher feedback. The chapter is structured around the key research questions, the architecture and design of the platform, the procedure for classroom testing, and the evaluation framework used to assess the platform's performance, usability, and educational value.

## 3.1 Research questions

The central research question guiding this project is:

> *"Can an Elo rating system accurately predict a student's proficiency level and provide optimally challenging questions?"*

This question is further explored through the following sub-questions:

- How well does the Elo-based model match student proficiency to question difficulty?
- Can adaptive difficulty encourage engagement and progression in learning?
- How do students and teachers perceive the usability and educational value of the system?

It is hypothesized that the Elo-based adaptive learning system will effectively match question difficulty to student ability, leading to increased engagement and accurate measurement of proficiency over time. Furthermore, it is expected that both students and teachers will find the system intuitive and pedagogically valuable.

## 3.2 System design

The system is designed as a full-stack web platform with separate interfaces for students and teachers. It uses an Elo-style rating engine to dynamically assign question difficulty based on student performance. Questions are either fetched from a database or generated in real time using the OpenAI API, depending on the student's current rating and the available question pool.

The platform is structured into the following core components:

- **Student Interface:** Allows students to log in, complete multiple-choice quizzes, and track their performance across five math topics.
- **Elo Rating Engine:** Calculates and updates both student and question ratings after each response.
- **Question Generator:** Uses the OpenAI API to generate new quiz questions when none match a student's Elo within a specified range.
- **Teacher Dashboard:** Provides an overview of student progress and question metrics, and allows manual moderation of generated content.
- **Database:** Stores user data, quiz interactions, question metadata, and Elo ratings using Firebase Firestore.

A visual diagram of the platform architecture is provided in Figure 3.1, which illustrates how data flows between components and how adaptive behavior is driven by real-time updates to student and question ratings.

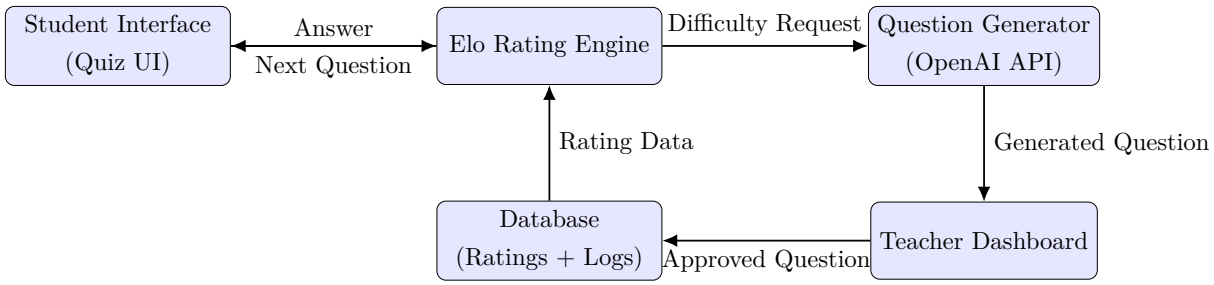


Figure 3.1: System architecture of the adaptive learning platform.

## 3.3 Justification for design decisions

The design of *EloQuiz* was shaped by a combination of pedagogical goals, technical feasibility, and user-centric principles relevant to middle school learners. Below, the key components of the system are justified with respect to their intended function and alignment with the overall research objectives.

**Use of the Elo rating system** The Elo rating system was selected as the foundation for adaptive difficulty due to its simplicity, interpretability, and proven track record in competitive environments like chess. Unlike more complex Bayesian models, Elo offers fast updates, minimal tuning requirements, and an intuitive score scale that can be easily visualized and interpreted by users.

**AI-generated question content** To populate the question pool dynamically, GPT-based content generation was used. This decision was motivated by the need for scalable and flexible question authoring, especially in a system that adapts to a wide range of skill levels. While human-curated questions offer higher reliability, AI generation allows coverage of multiple topics and difficulty bands with minimal manual overhead. A manual post-generation review mechanism was added to ensure content quality.

**Separation of student and teacher interfaces** Distinct interfaces for students and teachers were implemented to accommodate their differing needs. Students require a focused, minimal interface optimized for engagement and clarity during quiz sessions. Teachers, on the other hand, benefit from detailed analytics and control over question moderation and student linkage.

**Firebase and google cloud infrastructure** Firebase was selected for its integration of authentication, real-time database capabilities, and ease of deployment. Paired with Google Cloud Run for scalable backend execution, this infrastructure choice enabled quick iteration and smooth handling of classroom-level concurrency. The use of Docker further streamlined deployment and testing in different environments.

**Topic-specific Elo ratings** Instead of assigning a single Elo score per student, the system tracks separate Elo ratings for each math topic. This allows for more nuanced understanding of proficiency and better targeting of question selection. A student strong in algebra but weak in geometry can be challenged accordingly.

**Minimal instructional overhead** Given the age of the target users, the system was intentionally designed to be usable with minimal guidance. Buttons were labeled with simple language, navigation flows were kept linear, and the visual design avoided unnecessary complexity. The hypothesis was that a frictionless onboarding process would result in higher engagement and more authentic behavioral data during testing.

**Live feedback and streak tracking** Immediate feedback and streak visualization were added to tap into students' extrinsic motivation. These game-inspired mechanics

are intended to increase session time, reinforce correct behavior, and make the experience more engaging without detracting from educational value.

Overall, these decisions reflect a pragmatic balance between educational theory, technical efficiency, and user experience, prioritizing accessibility and adaptability over theoretical perfection.

## 3.4 User testing

**Purpose** The primary purpose of the user testing was to evaluate the functionality, usability, and educational relevance of the adaptive learning platform in a real classroom setting. While the technical system was developed and tested in isolated conditions during implementation, the user test aimed to investigate how the platform performs when used by actual students and teachers under normal school conditions.

Specifically, the goal was to observe how fifth-grade students interact with the quiz interface, whether the Elo-based difficulty adjustment works as intended in practice, and how intuitive the platform is for new users without technical instruction. Additionally, feedback from the teacher perspective was supposed to be collected to assess the usefulness of the learning analytics features and the practical feasibility of adopting such a system in everyday teaching.

The user test serves both as a qualitative validation of the system's design and as an opportunity to uncover usability issues, performance bottlenecks, or unintended behaviors that may not have emerged during internal testing. It also provides insights into user experience and engagement factors that are critical for any educational technology intended for classroom use.

**Participant overview** The user test was conducted in collaboration with Krebs' Skole, an independent school located in Copenhagen. Two fifth-grade classes participated in the study, comprising a total of approximately 40 students aged 11–12 years old. All participants were regular students with varying levels of mathematical proficiency and no prior exposure to the platform or its interface.

The test was facilitated by the school's mathematics teacher, who was also present during the session to provide classroom support and context. Prior to the test, the teacher was informed about the structure and goals of the platform, and was given access to the teacher dashboard to explore its features.

No personal data was collected during the test. Each student used a randomly generated user ID for login, and all responses were anonymized. The primary aim was not to evaluate individual student performance, but rather to assess system usability and engagement in a realistic classroom setting.

**Test procedure** The test was conducted during a standard 45-minute mathematics lesson. The session began with a short verbal introduction to the platform, including a demonstration of the login process and a brief explanation of the system's adaptive question logic. Each student then logged in to the platform using a unique user ID and connected to the teacher's profile via a four-digit classroom code.

Students were instructed to answer as many multiple-choice math questions as they could within a 20–25 minute time frame. The questions were dynamically adjusted in difficulty based on student performance, using an Elo-based rating system for each of the five predefined math topics.

Throughout the session, the system automatically recorded question responses, estimated difficulty levels, and individual Elo progression for each topic. The teacher had access to real-time analytics through the dashboard, showing aggregated class performance and question statistics.

At the end of the session, students were asked to complete a short feedback questionnaire via Google Forms, focusing on their experience with the platform, clarity of the interface, and perceived difficulty of the questions.

Table 3.1: User Testing Procedure Overview

| **Step** | **Description** |
|---|---|
| Introduction | The test facilitator introduced the purpose of the session and briefly explained the platform to the students and teacher. |
| Login | Students logged in using a unique user ID and joined the teacher's class via a 4-digit code. |
| Quiz Session | Students answered as many adaptive multiple-choice math questions as possible within a 20–25 minute session. Question difficulty adjusted in real time based on performance. |
| Teacher Monitoring | The teacher used the dashboard to monitor class performance and view live data such as Elo scores and answer accuracy. |
| Feedback Collection | Students completed a short feedback form about their experience. |
| Wrap-Up | The session ended with a brief discussion and closing remarks. No personal data was collected during the test. |

**Data collection** The data collected during the user test served both quantitative and qualitative purposes, aimed at evaluating the platform's usability, adaptivity, and perceived value in an educational setting.

**1. System-generated data:** As students interacted with the quiz system, key data points were automatically recorded and stored via Firebase/Firestore. This included:

- Student responses to each multiple-choice question
- Elo score progression per topic
- Correct/incorrect answer ratios

These metrics enabled analysis of how the adaptive question algorithm performed in practice and whether difficulty calibration functioned as intended.

**2. Student feedback:** After the quiz session, each student completed a short anonymous questionnaire via Google Forms. The form focused on:

- Overall user experience
- Perceived question difficulty
- Clarity of the interface
- Motivation and engagement

This provided insight into how students experienced the platform beyond the data logs.

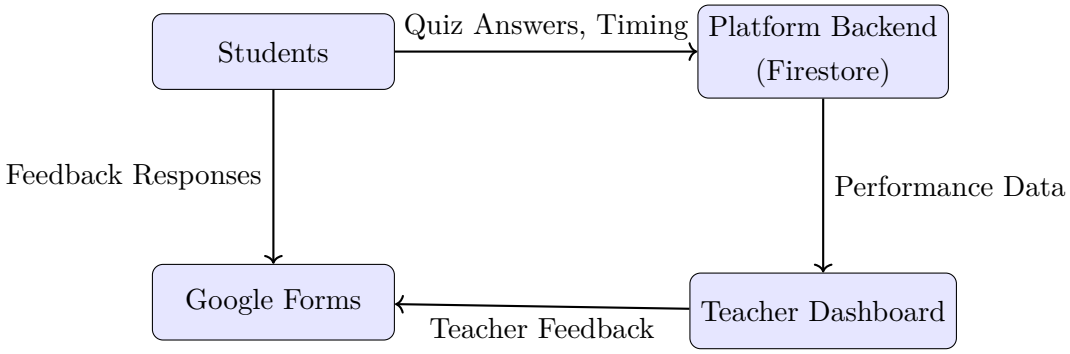


Figure 3.2: Overview of data collection sources during user testing.

## 3.5 Evaluation criteria

To assess the effectiveness and usability of the adaptive learning platform, several evaluation criteria have been defined. These criteria are aligned with the goals of the system: to deliver personalized, appropriately challenging math questions and provide meaningful feedback to both students and teachers.

**Accuracy of difficulty matching** This criterion evaluates whether the system successfully adapts question difficulty to match student proficiency. It was assessed by comparing students' Elo ratings with the Elo values of the questions they receive. A good match is indicated by question ratings that stay within a reasonable range (e.g., $\pm 200$ Elo) of the student's rating.

**Response accuracy and progression** Student performance was tracked to see whether the platform adjusts difficulty over time in a way that encourages learning. Metrics include improvement in accuracy, reduced response times, and smooth transitions between difficulty levels. A positive trend may indicate that the system is offering appropriately leveled challenges.

**System usability and user satisfaction** Qualitative feedback from students was used to evaluate usability. This includes whether the interface was intuitive and whether students felt engaged. Data was drawn from structured questions in post-test Google Forms surveys.

**System reliability and latency** The platform's ability to respond promptly and remain stable under simultaneous usage is also evaluated. During the live user test, backend logs and observational notes were used to document any downtime, delays, or bugs encountered.

## 3.6 Limitations

While this study aimed to evaluate the effectiveness of an Elo-based adaptive learning platform, several limitations should be noted.

**Limited sample size** The user testing was conducted in a controlled environment with a small number of classes and students. While the feedback and performance data are valuable, the relatively narrow demographic and sample size limit the generalizability of the findings.

**Short-term evaluation** Due to time constraints, students only interacted with the platform during a single session. As a result, it was not possible to assess long-term learning effects, engagement over time, or how well the system adapts as students progress through extended learning cycles.

**Data quality and noise** The Elo rating model depends on frequent and varied student responses to adjust difficulty accurately. With limited interaction and occasional guessing

behavior, the ratings may not always reflect true proficiency. Additionally, feedback collected through surveys may suffer from response bias or lack of detail.

**Model output reliability** Question generation relies on a large language model (GPT), which occasionally produced ambiguous or flawed questions despite efforts to constrain the format. This introduces a level of uncertainty into the quality of the content and may have affected the learning experience.

**Teacher involvement** Although the platform includes features for teacher review and oversight, the effectiveness of these tools were not deeply evaluated. Future studies should assess how easily teachers can integrate the system into their workflow and how they perceive its utility.

## 3.7 Ethical considerations

The development and testing of the adaptive learning platform were guided by core ethical principles related to research with human participants, especially in educational settings involving minors.

**Informed consent** All participants, including students and teachers, were informed about the purpose of the study, the nature of their involvement, and their right to withdraw at any time. For minors, participation was contingent upon approval from the school and teachers, who acted as proxies for parental consent within the school's established policy.

**Data privacy** User data collected during testing including quiz responses, performance metrics, and feedback was stored securely. Students were told to just use their first names when signing up for the website. However it was considered that this could give some confusion if multiple students had the same first name, when the teacher had to go through student statitics. Data was only used for the purposes of this study and was not shared with any third parties.

**Use of AI-generated content** The platform uses OpenAI's API to dynamically generate quiz questions. Efforts were made to validate the accuracy and appropriateness of these questions before exposing them to students. A review system was also in place to allow teachers to monitor and remove problematic questions.

**Bias and fairness** The adaptive model used an Elo rating system intended to deliver a personalized and fair challenge for each student. However, the risk of implicit bias in AI-generated content and performance interpretation was acknowledged. Future iterations of the system should incorporate more robust fairness checks and content validation pipelines.

**Non-harmful intent and EU AI Act considerations** The platform was developed with the clear intention of enhancing learning and engagement, not to replace teacher judgment or deliver formal academic assessments. During classroom use and testing, teachers retained full oversight, and all data collection was conducted transparently and minimally.

In line with the principles outlined in the EU AI Act, particular care was taken to avoid exploiting the vulnerabilities of users due to age—in this case, children. Article 5 of the Act prohibits the use of AI systems that manipulate or exploit such vulnerabilities, especially in ways that could cause physical or psychological harm ( Foundation [2024]). EloQuiz was explicitly designed to support educational interaction, not to automate decisions or profiling, and makes no behavioral predictions that affect student treatment, access to resources, or academic standing. It provides quiz questions tailored by a lightweight Elo model, primarily for engagement and self-assessment, not academic evaluation.

Furthermore, the EU AI Act classifies AI systems used in education and vocational training as potentially high-risk if they influence student progression, assessment, or behavior monitoring ( Service [2025]).

To align with the Act's transparency requirements, interactions with the platform during testing were overseen by a teacher. The design philosophy adheres to *human-in-the-loop* principles, ensuring educators remain central to the learning process ( SoBigData [2024]).

# 4 Implementation

This chapter details the technical implementation of the adaptive learning platform, EloQuiz. It covers the development environment setup, the architectural choices, and the specific technologies used to build the system. The implementation integrates a responsive frontend built with Vue.js to create a seamless user experience and a robust backend developed using Flask for reliable performance. Emphasis is placed on clearly delineating frontend and backend components to ensure modularity, maintainability, and scalability. This chapter also provides a comprehensive overview of the Elo-based adaptive logic, the dynamic question generation via the OpenAI API, and the data storage schema utilizing Firebase. Additionally, deployment strategies and tools employed for version control and testing are discussed, alongside identified limitations and known issues encountered during development.

## 4.1 The setup

**Frontend setup** To initialize the frontend portion of the project, a new folder named EloQuiz was created using Visual Studio Code. The Vue project was then scaffolded using the Vue CLI with the following terminal commands:

```
npm install -g @vue/cli
vue create frontend
```

During setup, the default configuration was selected: Default ([Vue 3] babel, eslint).

Next, the following commands were executed to install dependencies and start the development server:

```
cd frontend
npm install
npm run dev
```

The cd frontend command navigates into the project directory. The npm install command reads the package.json file and installs all required dependencies, creating a

node_modules folder and generating or updating the package-lock.json file to ensure consistent version control.

Finally, npm run dev launches the development server. This server provides live reloading and hot module replacement, enabling the developer to preview changes in real-time. It is the primary command used to run and test the frontend during development.

From here it is possible to create views and compenents in the src folder. An overview of these will be given in the System Architecture Overview section.

**Backend setup** The backend was developed using the Flask web framework. A dedicated folder named backend was created to contain all relevant files, including route definitions, authentication logic, API endpoints, and configuration files. The structure includes key files such as main.py, auth.py, quiz.py, statistics.py, and a requirements.txt file to manage dependencies.

The backend environment was set up using pip, Python's package manager. First, a virtual environment was created (optional but recommended), and the necessary packages were installed by running:

```
pip install -r requirements.txt
```

The requirement file contained the following:

```
Flask==3.0.3
gunicorn==21.2.0
flask-cors==4.0.0
firebase-admin==6.5.0
python-dotenv==1.0.1
openai==1.30.1
google-cloud-secret-manager==2.19.0
```

To start the backend during development, the following command was used:

```
python main.py
```

This command runs the Flask application, which is defined and initialized within main.py. The backend exposes a set of API endpoints that handle user authentication, quiz logic, statistics aggregation, and database interactions with Firebase.

## 4.2 Frontend architecture overview

The adaptive learning platform is designed as a modular web application that delivers personalized mathematics quizzes to students while enabling teachers to monitor progress.

The system integrates a frontend user interface, a backend logic engine, a database layer, and external services for dynamic question generation.

The frontend of the adaptive learning platform is implemented as a single-page application (SPA) using Vue.js, providing a responsive and intuitive interface for both students and teachers. The user experience is tailored to each role, enabling seamless interaction with the backend API.

**Frontend views and components** The platform provides two user-facing interfaces: a student-facing quiz UI and a teacher-facing analytics dashboard. The student UI supports interactive multiple-choice quizzes and shows topic-level Elo ratings. Teachers can view students' performance metrics and quiz questions via the dashboard.

The frontend consists of the following central views. The remaining views are in the appendix.

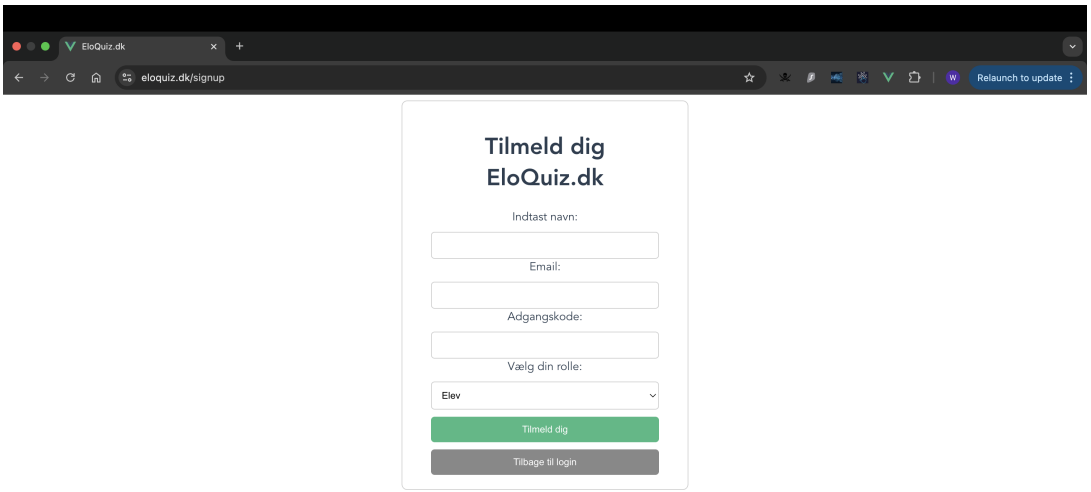


Figure 4.1: A screenshot of UserSignup.vue

**UserSignup.vue** The UserSignUp.vue component provides the registration interface for new users of the application. It allows individuals to create an account by submitting a name, email, password, and selecting a role either student or teacher.

The form is implemented using Vue's v-model directive to bind form inputs to component data properties. Upon submission, the signUp() method is triggered. This method carries out several key operations:

1. User creation: The component uses Firebase Authentication's createUserWithEmailAndPassword method to register the user with the provided email and password.

2. User profile storage: After successful authentication, additional user data, including name, email, role, and an initial current_streak value, is stored in Firestore under the users collection using the user's unique ID as the document key.

3. Automatic login and redirection: The user is automatically logged in via signInWithEmailAndPassword immediately after registration. Based on the selected role, the user is redirected to either the /teacher-dashboard or /student-dashboard route.

4. Error handling: If any part of the process fails, an alert is shown to the user displaying the relevant error message returned by Firebase.

The component also provides a secondary button labeled "Tilbage til login," which allows users to navigate back to the login screen if they already have an account.

Visually, the interface is designed with simplicity and usability in mind. All form fields and buttons are consistently styled and responsive, providing a seamless onboarding experience for both students and teachers.

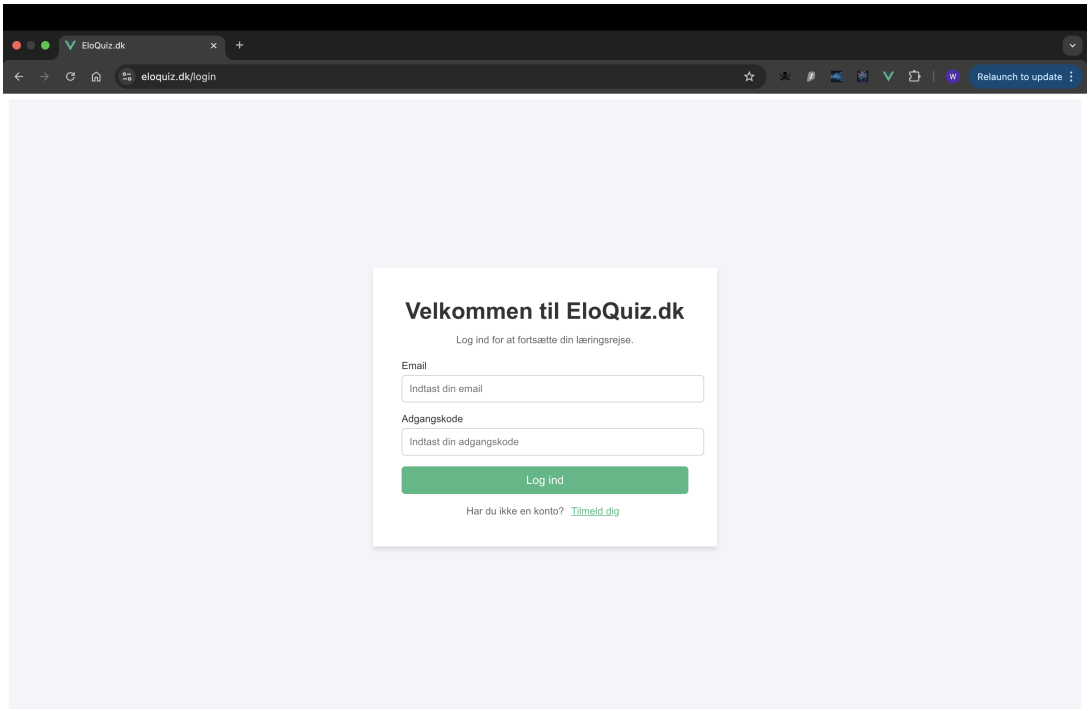


Figure 4.2: A screenshot of UserLogin.vue

**UserLogin.vue** The UserLogin.vue view serves as the primary authentication gateway for all users students, teachers, and administrators. It presents a login form that accepts email and password credentials and authenticates users via Firebase Authentication.

Upon submission, the login() method is triggered. This method:

1. Calls Firebase's signInWithEmailAndPassword to verify the user's credentials.

2. If authentication is successful, retrieves the user's document from the Firestore users collection using the user's unique ID.

3. Extracts the user's role field and redirects the user to the appropriate route:

- /student-dashboard for students
- /teacher-dashboard for teachers
- /admin-panel for administrators

If the user's document is not found or the role is invalid, an error is shown to the user.

The component includes UI feedback mechanisms such as a loading state (which disables the login button and displays a loading message during processing) and error alerts for failed logins. It also includes a link to the sign-up page for users who do not yet have an account.

Overall, UserLogin.vue provides a secure and user-friendly entry point to the application, while enforcing role-based routing logic essential to the platform's adaptive structure.

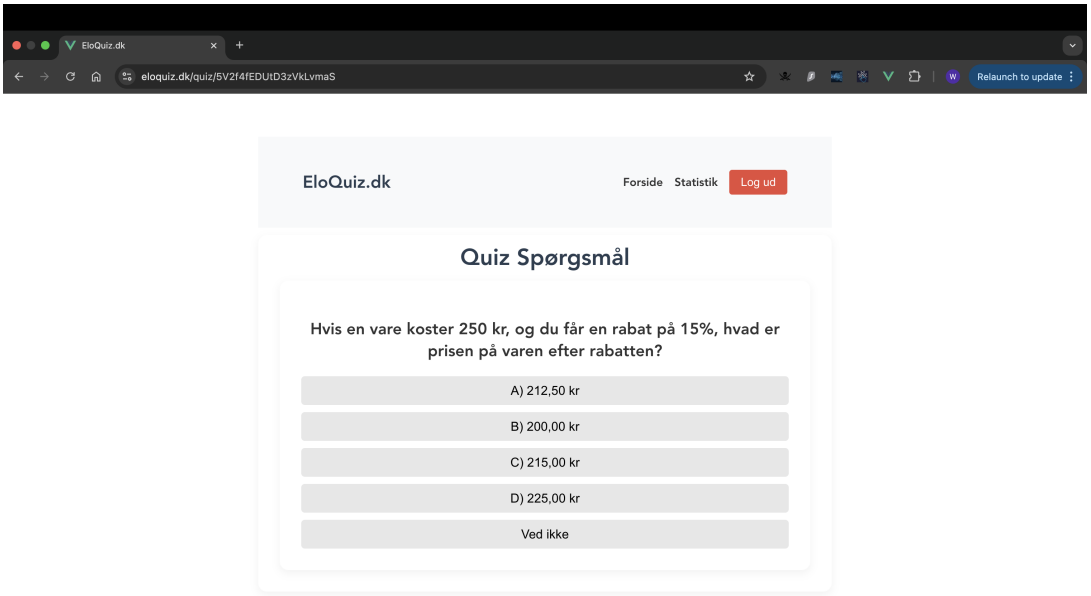


Figure 4.3: A screenshot of QuizScreen.vue before answering a question

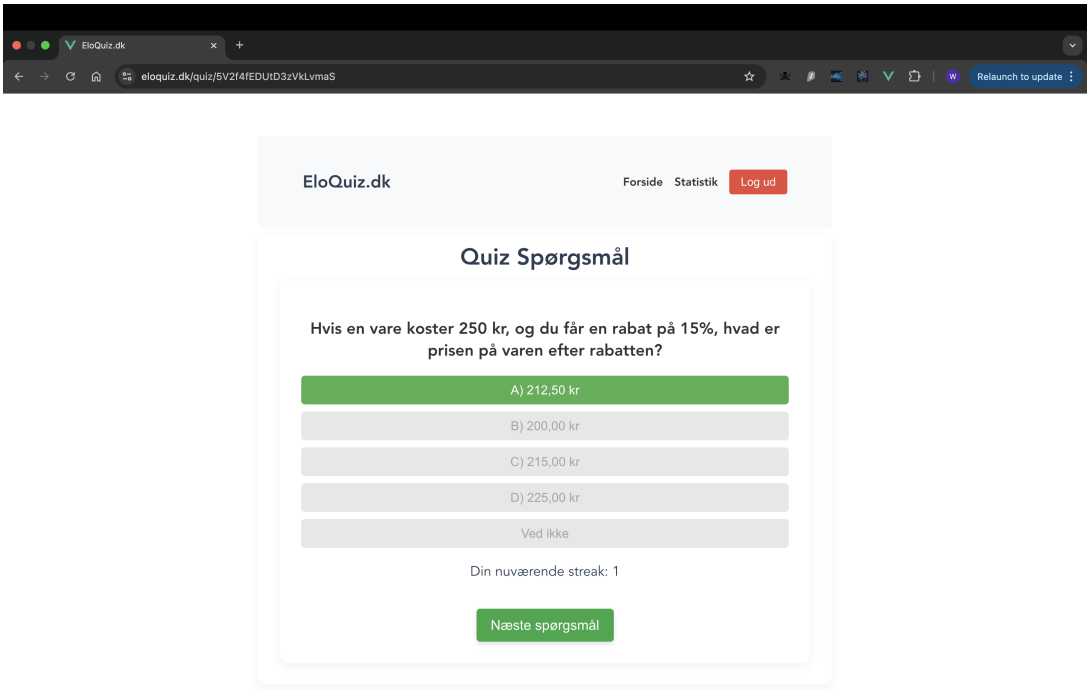


Figure 4.4: A screenshot of QuizScreen.vue after answering a question

**QuizScreen.vue** The QuizScreen.vue view is the central interface through which students interact with multiple-choice questions. It is a core part of the student-facing user interface and is responsible for rendering individual quiz questions, managing user input, and providing real-time feedback.

Upon loading, the component fetches the relevant question data from the backend using the question ID passed via the route parameters. It displays the question text along with a list of possible answer options, including a "Ved ikke" ("I don't know") fallback to reduce random guessing. When a student selects an answer, the interface provides immediate visual feedback: the selected option is highlighted as correct or incorrect, and the actual correct answer is shown if the response was incorrect.

After answering, the student is presented with their current correct streak, a motivational element designed to reinforce engagement and consistency. A "Next Question" button allows the user to proceed, triggering a new fetch for the following question and repeating the process.

Overall, QuizScreen.vue acts as the interactive engine of the quiz system, balancing usability, educational feedback, and backend communication.

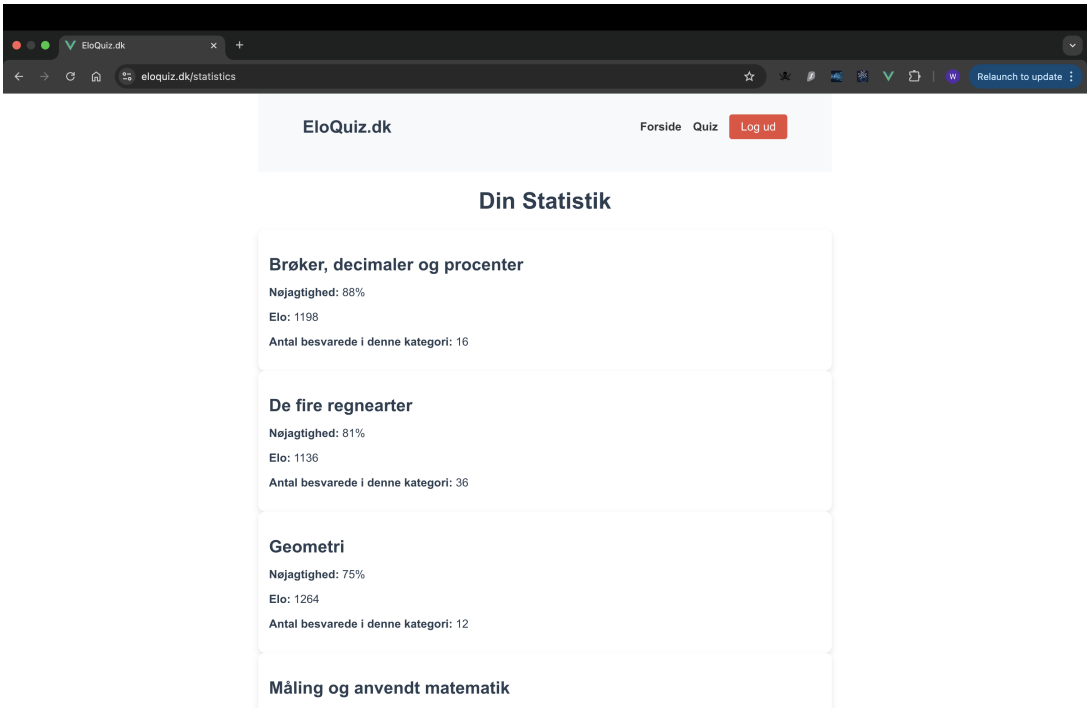


Figure 4.5: A screenshot of StatisticsPage.vue

**StatisticsPage.vue** The StatisticsPage.vue view displays a student's personal performance data across various mathematical topics. It is part of the student-facing interface and is designed to provide immediate, topic-specific feedback based on the student's quiz history. The view is only accessible to logged-in users and dynamically loads data from the backend upon component mount.

When the component initializes, it checks for a currently authenticated user using Firebase Authentication. If no user is logged in, an error message is shown. If authentication is successful, it makes a GET request to the backend's /api/get_user_stats endpoint, passing the user's unique ID as a query parameter.

The backend responds with a structured JSON object containing performance metrics for each topic the student has attempted. These include:

- **Accuracy**: The percentage of correct answers.
- **Elo rating**: A skill rating that adapts based on performance.
- **Number of answered questions**: Total interactions within that topic.

The component processes this data and renders a card for each topic. Each card displays the topic name, accuracy, Elo rating (rounded), and total questions answered. If no data is available, i.e., the student has not yet answered any questions, a fallback message is displayed encouraging participation.

A loading state is shown while statistics are being fetched, and error messages are conditionally rendered if the fetch fails. The component also includes a navigation bar and styling consistent with the rest of the student interface.

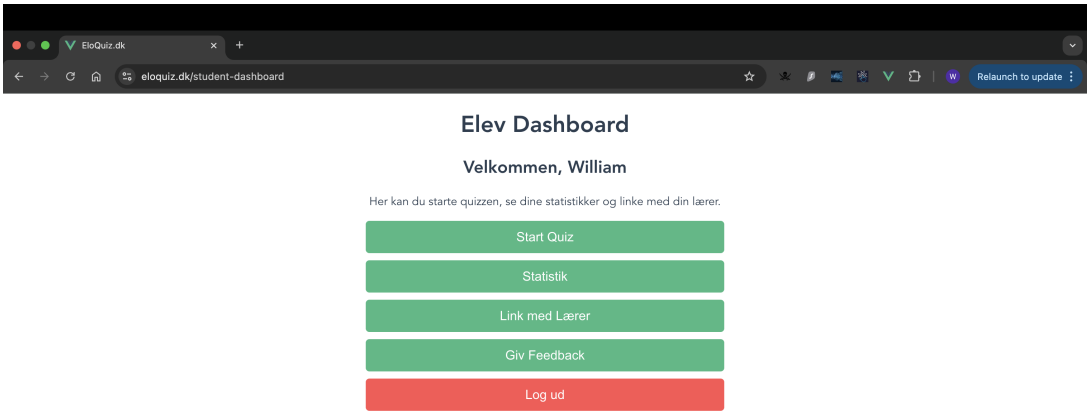


Figure 4.6: A screenshot of StudentDashboard.vue

**StudentDashboard.vue** The StudentDashboard.vue view serves as the central hub for student interaction within the application. It is the landing page presented to authenticated student users after login and provides access to core features such as starting quizzes, viewing statistics, linking to a teacher, providing feedback, and logging out.

Upon mounting, the component checks for an authenticated user using Firebase Authentication. If a user is logged in, the component retrieves the corresponding document from the Firestore database's users collection to load the user's display name and other associated metadata. This personalized information is then used to greet the student within the interface.

The component offers the following functionalities:

- **Start Quiz**: Initiates the quiz experience by requesting the next appropriate question from the backend using the student's user ID. The student is then redirected to the corresponding QuizScreen.vue route with the question ID.
- **View Statistics**: Redirects the student to the StatisticsPage.vue component, which visualizes their performance metrics across various topics.
- **Link with Teacher**: Opens a modal where the student can enter a teacher code. Upon confirmation, this code is saved to the student's document in Firestore, allowing the student to be associated with a specific teacher account.
- **Give Feedback**: Opens a Google Form in a new tab, allowing students to submit qualitative feedback about their experience.
- **Log Out**: Signs the student out using Firebase Authentication and redirects them

to the login page.

Visually, the component is styled to maintain consistency with the rest of the user interface and includes modal logic for teacher code submission. All interactive buttons are designed with responsiveness and accessibility in mind.

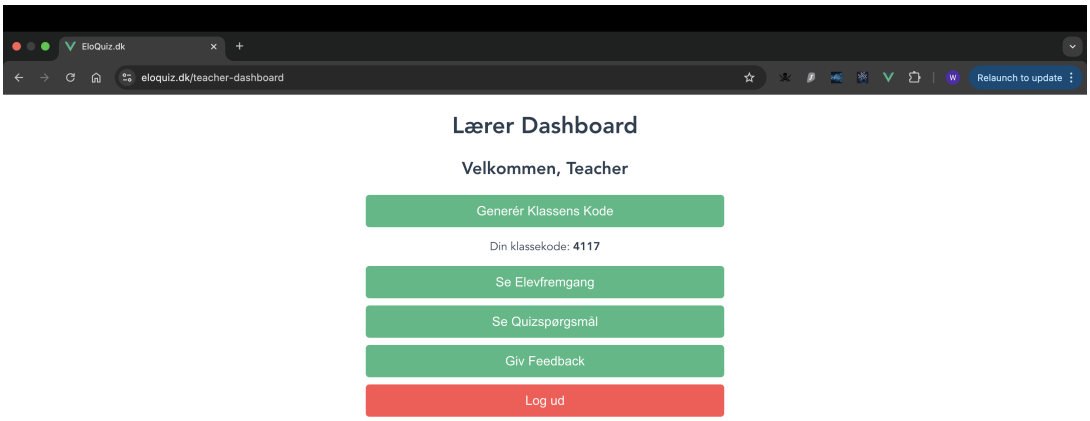


Figure 4.7: A screenshot of TeacherDashboard.vue

**TeacherDashboard.vue** The TeacherDashboard.vue view serves as the main interface for users logged in as teachers. It provides a centralized overview and access to teacher-specific functionality, including student management, quiz oversight, and administrative tasks.

Upon mounting, the component verifies the authenticated user's status using Firebase Authentication. It then retrieves the teacher's profile data from Firestore, displaying a personalized welcome message that includes the user's name when available.

The dashboard provides the following core functionalities:

- **Generate Class Code**: Teachers can generate a unique numeric class code used by students to link their accounts. If a class code already exists in the Realtime Database, the system prevents generating a duplicate and instead displays the current code.
- **View Student Progress**: Redirects the teacher to the StudentProgress.vue view, where individual student performance data can be reviewed by topic.
- **View Quiz Questions**: Navigates to the ViewQuestions.vue route, providing an overview of the available quiz questions stored in the system.
- **Give Feedback**: Opens a Google Form in a new browser tab, allowing teachers to

provide user feedback on the system.

- **Log Out**: Signs the user out using Firebase Authentication and redirects them to the login screen. Proper error handling is included to ensure smooth operation.

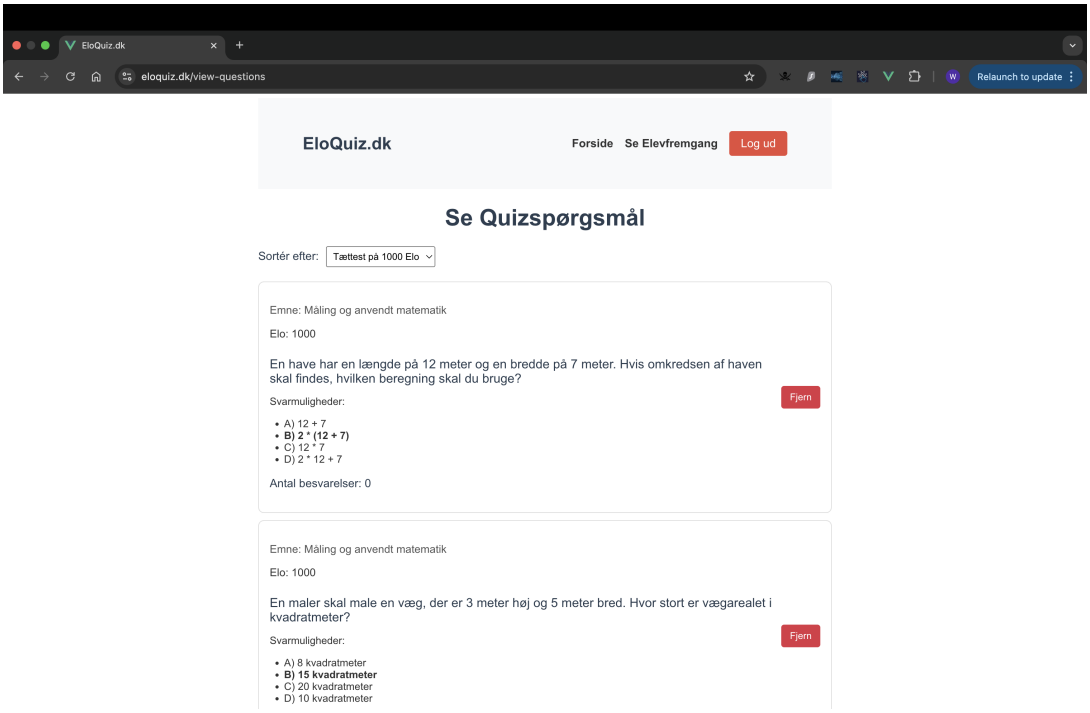


Figure 4.8: A screenshot of ViewQuestions.vue

**ViewQuestions.vue** The ViewQuestions.vue view is part of the teacher-facing interface and is designed to display a list of all quiz questions currently stored in the system. It provides a streamlined overview for reviewing question content, Elo ratings, topic categorization, and answer options. This functionality is essential for monitoring the quality and distribution of questions within the adaptive learning model.

Upon mounting, the component fetches question data from the backend endpoint /api/get_questions. Each question object includes metadata such as the question text, associated topic, current Elo rating, total number of times the question has been answered, and a list of multiple-choice options. The correct answer is visually emphasized to distinguish it from distractors.

The interface also includes a sorting feature, allowing teachers to organize questions based on Elo rating:

- Closest to 1000 (neutral difficulty),
- Most answered,
- Lowest to highest Elo, or
- Highest to lowest Elo.

This helps educators quickly identify outliers or miscalibrated questions.

Teachers can also delete questions by clicking a “Fjern” button, which triggers a DELETE request to /api/remove_question. This enables real-time content moderation and supports iterative question refinement based on performance or feedback.

Overall, ViewQuestions.vue provides teachers with direct oversight of the question bank and plays a critical role in maintaining the integrity and balance of the adaptive system's question pool.

**Remaining views and components** The remaining views are AdminPanel.vue, RedirectHandler.vue, StudentLink.vue, StudentProgress.vue and NavBar.vue. Details can be found in the appendix.

## 4.3 Backend architecture overview

The backend of the EloQuiz system is built using the Flask web framework in Python and is designed to provide an API interface for client applications. It handles core functionalities such as user authentication, quiz logic, adaptive difficulty calculations, and performance tracking. The backend is structured into modular components, each responsible for a specific domain of functionality, and is organized using Flask Blueprints to maintain scalability and separation of concerns.

Firebase is used as the primary backend-as-a-service solution, leveraging both Firestore and Realtime Database for data storage, along with Firebase Authentication for secure user management. Configuration and sensitive credentials are managed through environment variables loaded from a .env file, and the Firebase Admin SDK is initialized at startup using a service account key.

The entry point for the application is the **main.py** file, which initializes the Flask app, sets up CORS headers, registers the relevant blueprints, and starts the server. The application exposes multiple API endpoints grouped across several modules:

- **auth.py** for login and user authentication logic
- **quiz.py** for serving and evaluating quiz questions
- **statistics.py** for aggregating and serving user performance metrics

The following sections provide an in-depth explanation of each backend module, their key functions, and how they interact with the system as a whole.

**main.py – Application entry point** The `main.py` file serves as the entry point for the backend application. It is responsible for initializing the Flask server, loading configuration settings, and registering the system's modular components using Flask Blueprints.

**Key responsibilities:**

- **Environment Configuration:** The application begins by loading environment variables from a `.env` file using `python-dotenv`.
  This includes the `GOOGLE_APPLICATION_CREDENTIALS` path used to authenticate with Firebase. If the service account file is missing or incorrectly configured, the program raises an error to prevent the server from starting without proper credentials.
- **Firebase Initialization:** The Firebase Admin SDK is initialized using the service account key, with the Realtime Database URL configured explicitly. This allows the backend to interact with both Firestore and the Realtime Database for user data, statistics, and quiz metadata.
- **Blueprint Registration:** To maintain a clean project structure and logical separation of concerns, different parts of the API are modularized into separate files using Flask Blueprints:
  - `auth_bp` handles authentication logic and is imported from `auth.py`.
  - `quiz_bp` handles question serving and submission, defined in `quiz.py`.
  - `stats_bp` provides access to performance metrics, defined in `statistics.py`.
- **CORS Configuration:** The Flask-CORS extension is enabled to allow cross-origin requests from the frontend, which is served separately (e.g., via `npm run dev` on another port during development).
- **Health Check Endpoint:** A root route (`/`) is defined to return a simple confirmation message, allowing developers to verify that the backend server is running correctly.
- **Development Server Startup:** When the file is run directly, the Flask application starts in debug mode, listening on `0.0.0.0` at port `5001`. This configuration allows the server to be accessed from other devices on the local network during testing.

## auth.py – User authentication and registration

The `auth.py` module handles user signup, authentication, and system health checks. It defines a Flask Blueprint named `auth_bp` and includes routes for creating new user accounts, verifying server status, and (optionally) handling login logic.

**Key responsibilities:**

- **User signup – `/api/signup`**
  This endpoint allows users to create a new account. It expects a JSON payload containing an email, password, optional name, and role (defaulting to "student" if not provided). The backend:
  - Creates the user in Firebase Authentication using the provided credentials.
  - Initializes a corresponding user document in Firestore, storing metadata including name, email, role, current streaks, and a default Elo rating of 1000 across all topics.
  - Returns a success message and the new user's unique ID.

  If any step fails (e.g., email already in use, invalid password), the server responds with an error message and status code 400.

- **Login – `/api/login`**
  This placeholder endpoint is currently stubbed and returns a basic message. Authentication in the application is handled client-side using Firebase Authentication directly from the frontend.

- **Health check – `/api/health`**
  A lightweight GET endpoint used to confirm the backend is operational. Returns a simple status message ("OK") and HTTP 200 status code. This is useful for monitoring tools or local development.

- **Firebase initialization:** If no Firebase app is currently running, the module initializes Firebase using the service account credentials defined in the environment variables. A Firestore client instance is then made available for reading and writing user data.

- **CORS configuration:** Cross-Origin Resource Sharing (CORS) is enabled specifically for the authentication blueprint, allowing frontend applications hosted on a different origin to interact with these endpoints.

Overall, `auth.py` establishes the foundational logic for account creation and backend availability. It plays a critical role in onboarding new users and maintaining secure, role-based access to the platform's core features.

**quiz.py – Quiz logic and adaptive functionality** The `quiz.py` module is responsible for the core functionality of the quiz system. It defines a Flask Blueprint named `quiz_bp` and exposes a series of API endpoints that manage the retrieval, generation, submission, and performance tracking of quiz questions. This module acts as the primary point of interaction between the frontend and the adaptive backend logic.

**Key responsibilities:**

- **Serving existing questions:** Endpoints such as `/api/get_question` and `/api/get_questions` allow the frontend to retrieve individual questions or fetch all questions stored in Firestore. Metadata such as topic, Elo rating, and answer count are included for each question.
- **Adaptive question selection:** The endpoint `/api/get_next_question` selects a question based on the student's performance history. It compares the student's Elo rating per topic to available questions, selecting the one with the closest Elo value. If no suitable question is available, a new one is generated using OpenAI's GPT model.
- **Question generation:** The backend uses the OpenAI API to generate new multiple-choice questions in Danish. The difficulty level is based on the student's current Elo in the topic. Questions are returned in JSON format and stored in Firestore with an initial Elo value based on the assigned difficulty.
- **Answer submission and elo update:** The `/api/submit_answer` endpoint processes student answers, records metadata (such as response time and correctness), and updates both the student's and question's Elo ratings using the Elo rating formula. It also tracks the student's streak and question history to avoid repetition.
- **Performance tracking:** Additional helper functions update student-specific performance metrics in Firebase, such as accuracy and average response time per topic. This data is used for both adaptation and visualization in the frontend.
- **Administrative utilities:** The endpoints `/api/reset_student_elo` and `/api/reset_question_elo` allow administrators to reset Elo ratings, either for a specific student or all questions. Answers related to reset questions are archived in Firestore.
- **Batch pre-generation of questions:** The `/api/pre_generate_questions` endpoint automates the generation of multiple questions across all topics and difficulty levels for testing or onboarding purposes.

Overall, `quiz.py` serves as the operational center for the backend's adaptive learning engine. It ensures that each student receives appropriately challenging content, while also maintaining a robust data record for evaluation and continuous improvement.

**statistics.py – Performance statistics and student management** The `statistics.py` module is responsible for computing and serving performance data for individual students and classrooms. It defines a Flask Blueprint called `stats_bp` and exposes several API endpoints that support teachers in tracking progress, viewing statistics, and managing student accounts.

**Key responsibilities:**

- **Get class statistics – `/api/get_students`**
  This endpoint accepts a teacher's user ID and retrieves the corresponding class code from the Firebase Realtime Database. It then queries Firestore for all users who have linked to that class. For each student, it aggregates quiz data from the `answers` collection using the helper function `compute_user_stats()`, and returns a structured JSON object containing:
  - Accuracy per topic
  - Current Elo rating per topic
  - Number of answered questions per topic
- **Get individual user statistics – `/api/get_user_stats`**
  This endpoint allows a logged-in user (typically a student) to retrieve their own performance metrics. It aggregates all answer data and merges the current topic-wise Elo ratings stored in their Firestore `users` document. The resulting object is used for personal feedback and visualization in the frontend.
- **Remove student – `/api/remove_student`**
  This endpoint allows a teacher to remove a student from their class. It deletes the user's document from Firestore and also purges their quiz answer history to maintain data integrity and privacy.

Overall, `statistics.py` provides critical backend support for the system's data-driven features. It enables both students and teachers to monitor progress, supports adaptive decision-making, and contributes to personalized learning paths through accurate performance feedback.

## 4.4 Elo rating system implementation

The core of the adaptive learning platform is the Elo rating system, originally developed to rank players in competitive games like chess. In this system, both students and questions maintain individual Elo scores per topic, which dynamically evolve based on performance. This allows for a personalized and continuously adapting quiz experience.

**Rating initialization** Each question is assigned an initial Elo score based on its labeled difficulty:

- Very Easy: 800
- Easy: 1000
- Medium: 1200
- Hard: 1400
- Very Hard: 1600

Students begin with a default Elo of 1000 for each topic.

**Matchmaking and question selection** When a student requests a new question, the system fetches all unanswered questions and selects one whose Elo is closest to the student's current rating in that topic. If the difference between the student and the closest available question exceeds a threshold (200 Elo points), a new question is generated via the OpenAI API. This ensures relevance and challenge alignment.

**Elo update mechanism** After answering, the system updates both the student's and the question's Elo using the standard Elo formula:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

$$R'_A = R_A + K \cdot (S_A - E_A)$$

Here, $R_A$ and $R_B$ are the Elo ratings of the student and the question respectively, $E_A$ is the expected score for the student, $S_A$ is the actual result (1 for correct, 0 for incorrect, and 0.25 for "Don't Know"), and $K$ is a constant (set to 32). The same logic is mirrored for updating the question's rating.

**Don't know option** To account for uncertainty, students may also select a "Don't Know" option. This is treated as a partial failure with a score of $S_A = 0.25$, ensuring that hesitation is penalized less harshly than an incorrect guess while still contributing to the model's learning.

**Streak tracking** To better understand user engagement, the system also tracks correct answer streaks, updating values such as `current_streak` and `longest_streak` on each submission.

**Result logging** Each interaction is stored in the Firestore database under a centralized `answers` collection, which records the question ID, answer correctness, topic, response time, and both the student and question Elo before update. These logs are used for performance analysis and system debugging.

## 4.5 Question generation via OpenAI API

To ensure continuous availability of suitable content tailored to the student's proficiency, the system integrates with the OpenAI API to generate new questions dynamically. This generation mechanism is invoked when no existing questions are within an acceptable Elo range of the student's current rating.

**Trigger conditions** If a student requests a new question and none of the available items are within 200 Elo points of their topic-specific rating, the system triggers the generation of a new question for that topic.

**Prompt construction** The generation process constructs a prompt that specifies:

- Topic (e.g., fractions, geometry)
- Difficulty level (Very Easy to Very Hard)
- Language constraint (Danish)
- Format requirements (JSON with question text, four options, and a correct answer)

**Model interaction** The system uses OpenAI's `gpt-4o` model via the API. It sends the constructed prompt and expects a response in strict JSON format. The prompt that were used to generate the questions was constructed in the following way:

```
Create a unique {difficulty}-level multiple-choice question about {topic}.
Use random numbers and vary the math operations involved to ensure the
questions are not repetitive.
Ask the questions in danish language.

Return ONLY valid JSON with the following structure (and nothing more):
{
    "question": "<Your question text>",
    "options": ["A) ...", "B) ...", "C) ...", "D) ..."],
    "answer": "<One correct option from the above>"
}
```

The JSON is parsed and validated. Any parsing error results in a fallback error response to the client.

**Metadata assignment** Generated questions are assigned:

- A topic label
- A difficulty label
- An initial Elo score based on difficulty

This metadata is stored alongside the question in Firestore for future retrieval and rating updates.

**Database insertion** Each generated question is added to the Firestore `questions` collection. Over time, this expands the question pool and reduces reliance on generation, especially for commonly requested Elo ranges.

## 4.6 Database schema and data flow

The platform uses Google Firestore to manage structured data related to users, questions, and answer logs. This section outlines the database schema and describes the data flow during typical user interaction.

**Firestore collections** The database is organized into the following key collections:

- `users`: Stores student data, including:
  - `elo` (map of topic-specific Elo ratings)
  - `question_answers` (list of answered question IDs)
  - `current_streak`, `longest_streak`
- `questions`: Contains all multiple-choice questions. Each document includes:
  - `question`, `options`, `answer`
  - `difficulty`, `elo`, `topic`
- `answers`: Stores logs of student answers with metadata:
  - `user_id`, `question_id`, `topic`
  - `response_time`, `correct`
  - `user_elo`, `question_elo` (at time of answer)
- `archived_answers`: Stores removed answer logs when resetting question Elo scores.

**Data flow overview** When a student interacts with the system, the following sequence occurs:

1. Student logs in and requests a new question via `/api/get_next_question`.
2. The backend fetches unanswered questions from the `questions` collection and selects the one closest in Elo to the student's rating.
3. If no close match exists, the system generates a new question using the OpenAI API and stores it in Firestore.
4. Upon answering, the student submits the result via `/api/submit_answer`.
5. The system updates the student's and question's Elo ratings using the rating formula, then logs the result in `answers`.
6. The question ID is appended to the student's `question_answers` list to prevent repetition.

This schema supports scalability, real-time updates, and user-specific adaptation without requiring heavy client-side state management.

## 4.7 Adaptive logic and question selection

The adaptive logic governs how questions are selected for each student based on their evolving skill level across predefined mathematical topics. This process ensures that each student receives problems that are appropriately challenging, fostering continuous engagement and learning progression.

When a student requests a new question via the endpoint `/api/get_next_question`, the backend retrieves the student's Elo ratings for each topic from the `users` collection. These ratings reflect the student's current proficiency and are updated dynamically after each answer submission.

Next, the system queries the `questions` collection to retrieve all questions the student has not yet answered. For each unanswered question, the system calculates the absolute difference between the question's Elo and the student's Elo in that topic. The algorithm identifies the subset of questions with the smallest Elo difference and randomly selects one from this set. If the closest Elo match is within a threshold (e.g., 200 Elo points), the selected question is returned.

If no suitable question exists within the threshold, the system dynamically generates a new question using OpenAI's API. This new question is tagged with an initial Elo based on its difficulty label (e.g., `"Easy"` corresponds to an Elo of 1000) and is stored in the database for future use.

This adaptive mechanism ensures the platform provides questions that closely match each student's level, leveraging real-time updates to personalize the learning experience.

## 4.8 Deployment

To ensure scalability, portability, and maintainability, the adaptive learning platform is deployed using Docker containers and hosted on Google Cloud Platform (GCP). This approach simplifies local development, testing, and production deployment.

**Docker containerization** The backend Flask application is containerized using Docker. This allows the entire application and its dependencies to be bundled in a consistent environment across different systems.

The Dockerfile defines a minimal Python-based image, installs required dependencies from a `requirements.txt` file, and exposes the appropriate port for serving API requests. A production-ready WSGI server such as Gunicorn is used to run the Flask application inside the container.

The use of Docker allows developers to spin up isolated instances of the application with a single command, significantly reducing environment-specific bugs and setup friction.

**Google cloud platform hosting** The containerized application is deployed to GCP using Cloud Run, a fully managed compute platform that automatically scales containers based on traffic. Cloud Run integrates seamlessly with Google Firestore, which is used as the backend database, and with Google Secret Manager for secure management of API keys and credentials.

The deployment pipeline is as follows:

1. The Docker image is built locally or via a CI pipeline and pushed to Google Container Registry.
2. Cloud Run is configured to pull the latest image and serve it with HTTPS.
3. Environment variables, including API keys and database credentials, are injected securely via Secret Manager.

**Deployment Script – `deploy.sh`** To streamline the deployment process, a custom Bash script named `deploy.sh` was created. This script automates the build and deployment of both the backend and frontend components to Google Cloud Run via the Google Cloud CLI.

**Key responsibilities:**

- **Backend Deployment:**
  - Builds a Docker image from the `./backend` directory using the Google Cloud project container registry path.
  - Pushes the image to Google Container Registry (GCR).
  - Deploys the image to Google Cloud Run using `gcloud run deploy`, specifying the platform, region (`europe-west1`), and setting `-allow-unauthenticated` access.
- **Frontend deployment:**
  - Builds the production-ready Vue frontend using `npm run build` with the `-prefix frontend` option.
  - Creates a Docker image from the built frontend code.
  - Pushes the image to GCR and deploys it to Cloud Run in the same way as the backend.
- **Error handling:** `set -e` is used at the beginning of the script to ensure that the script exits immediately if any command fails, preventing partial deployments.
- **Automation and Repeatability:** By encapsulating the deployment process in a single shell script, the process becomes both reproducible and less prone to human error. It also allows for quick redeployment during testing or iteration.

**Deployment commands overview:**

```
# Exit on error
set -e

echo "Building backend Docker image..."
docker build -t gcr.io/.../adaptivelearning-backend ./backend

echo "Pushing backend image to Google Container Registry..."
docker push gcr.io/.../adaptivelearning-backend

echo "Deploying backend to Google Cloud Run..."
gcloud run deploy adaptivelearning-backend ...

echo "Building Vue frontend..."
npm --prefix frontend run build
```

```
echo "Building Docker image..."
docker build -t gcr.io/.../adaptivelearning-frontend ./frontend

echo "Pushing to Google Container Registry..."
docker push gcr.io/.../adaptivelearning-frontend

echo "Deploying to Google Cloud Run..."
gcloud run deploy adaptivelearning-frontend ...
```

The full script ensures that both parts of the application are deployed consistently and are always based on the latest code and configuration.

## 4.9 Git and version control

Git and GitHub were used throughout the development process to manage version control and track progress. For the most part, this workflow functioned as intended. However, several challenges emerged during development.

Initially, the absence of a proper .gitignore file led to unintended commits, including sensitive files such as API keys. This caused push failures and required the regeneration of compromised keys. The issue was resolved by adding critical files such as .env and serviceAccountKey.json to the .gitignore and removing problematic commits from the repository history.

A more complex issue occurred when a detached HEAD state was inadvertently created. Unaware of its implications, a significant number of commits were made in this state, effectively isolating a large portion of the project's progress from the main branch. Attempts to reattach the detached HEAD and merge the changes resulted in conflicts and repository instability. Eventually, it became difficult to perform even basic operations such as committing or pushing changes.

Fortunately, a local backup of version 0.16 had been saved prior to the issue. This backup was used to initialize a new repository. As a result, the commit history from version 0.1 to 0.17 was lost, and the new repository begins with the restored state of version 0.16, followed by a reimplementation of version 0.17 and all subsequent updates.

## 4.10 Testing and debugging tools

During development, a combination of manual testing tools and real-time monitoring utilities were used to validate functionality and identify bugs across both the frontend and backend components of EloQuiz. While automated tests were not implemented in

this version of the project, a range of developer-focused tools were employed to ensure the system functioned correctly in a variety of edge cases.

**1. Postman and cURL – API Testing**

API endpoints were tested using Postman and curl. These tools allowed for sending custom HTTP requests to the Flask backend, verifying request/response structures, checking authentication behavior, and confirming that routes returned expected results for different user roles.

**2. Firebase console – database and auth debugging**

The Firebase Console was used extensively for monitoring and modifying Firestore documents and Realtime Database nodes. It also served as the primary interface for inspecting user accounts, verifying login behavior, and observing changes triggered by application actions (e.g., Elo updates, new question generation).

**3. Chrome DevTools – frontend debugging**

The browser's Developer Tools (Inspect Element) were used to debug frontend logic and layout issues. The Console tab was used to catch JavaScript errors and inspect variable states, while the Network tab allowed for tracing API calls, checking HTTP status codes, and verifying backend connectivity.

**4. Google cloud run logs – server-side monitoring**

Logs from Google Cloud Run were essential for diagnosing issues in the deployed backend. Console output from the Flask app was monitored via the GCP web interface, particularly for debugging failed requests, authentication errors, and uncaught exceptions in production.

**5. Manual test cases – end-to-end behavior**

Manual testing was conducted by simulating typical user scenarios across the three user roles. This involved creating test accounts, performing tasks such as answering quiz questions, linking students to teachers, and observing updates to statistics and Elo ratings. Admin features like resetting user data and question pools were also tested to ensure backend consistency. These end-to-end workflows helped confirm that each role-specific view functioned as expected within the application.

Although formal unit and integration testing were not part of this thesis scope, the combination of these tools provided a robust environment for identifying and resolving bugs throughout development and deployment.

## 4.11 Limitations and known issues

While the adaptive learning platform demonstrates effective functionality and scalability, several limitations and known issues should be acknowledged:

**Question quality and GPT generation errors** The platform relies on OpenAI's API to dynamically generate new math questions when no suitable ones exist in the database. Although the prompts are structured to ensure valid formatting, there are occasional failures in JSON parsing due to unexpected formatting or content hallucinations. This necessitates human review and filtering by teachers via the Teacher Review Interface, but introduces potential latency and inconsistency in content quality.

**Elo matching and question scarcity** If the system runs low on pre-generated questions near a student's Elo range, it may either select a question with a suboptimal difficulty match or resort to generating a new question on the fly. While this fallback mechanism ensures continuity, it can lead to uneven difficulty progression or delay in delivery.

**Scalability testing** Although the system is hosted using scalable infrastructure (Google Cloud Run), formal load testing with a large number of concurrent users has not been conducted. Therefore, the system's real-world performance under stress, e.g., a full school rollout remains to be evaluated.

**Simplified student modeling** The Elo system assumes that student ability and question difficulty can be modeled using a single scalar value per topic. This abstraction is efficient, but it oversimplifies real learning behavior and does not account for other cognitive factors such as guessing, fatigue, or partial knowledge.

**Frontend responsiveness** While the frontend functions well on desktop browsers, limited responsiveness testing has been done on tablets or smaller mobile devices. Some UI elements may require refinement for fully responsive design.

# 5 Results and evaluation

## 5.1 Test setup

An initial dataset of 178 math questions was generated using a combination of GPT-assisted authoring and manual refinement. Each question was reviewed to eliminate duplicates, incorrect answers, and structurally ambiguous or confusing phrasing. The finalized set covered five core topic areas and spanned a range of difficulty levels calibrated using Elo scores.

The original plan was to conduct testing with two separate 5th-grade classrooms (approximately 20 students each) during a standard 45-minute math lesson. However, due to a series of technical limitations and connectivity issues (described in the following section), only 9 students were ultimately able to participate in the sessions.

## 5.2 Technical difficulties during testing

During the first classroom test with a 5th grade class, the school's firewall initially blocked access to the EloQuiz website. To circumvent this, a temporary mobile hotspot was created using a personal phone. However, due to bandwidth limitations, only five students were able to connect simultaneously, restricting participation during the session.

In the second classroom session with another 5th grade class, the mobile hotspot was used again, this time supporting four students. Unfortunately, during this test, the application exceeded the free daily quota of 50,000 Firestore document reads. Since billing had not yet been enabled, the backend service became unavailable. Although the firewall restriction had been lifted by this point, the majority of the students were unable to use the platform due to the backend outage.

As a result of these issues, a follow-up user test was planned, however due to logistical challenges with the end of the school year, it was not possible to schedule another testing session, and therefore the following results consists of a smaller sample size than originally planned.

## 5.3 Results

**Quantitative data summary** The following table presents a summary of key metrics collected during the classroom testing sessions of EloQuiz. Between the two classroom a total of 9 students tested the program and answered a total of 230 questions, averaging to about 25.5 per student. From that comes the following statistics about each of the five categories of questions.

Table 5.1: Descriptive Statistics of Student Elo Ratings by Category (n = 9)

| **Category** | **Mean Elo** | **Std Dev** | **Min** | **Max** |
|---|---|---|---|---|
| Fractions, decimals and percentages | 1047.68 | 61.84 | 936.21 | 1163.25 |
| Geometry | 1049.68 | 45.18 | 950.62 | 1092.82 |
| Simple Algebra | 1048.84 | 26.61 | 1014.62 | 1118.40 |
| Measurement and applied mathematics | 1031.98 | 36.12 | 982.56 | 1096.83 |
| The four types of arithmetic | 1045.01 | 22.74 | 1014.69 | 1084.32 |

The table illustrates that average Elo scores across all five categories were fairly consistent, ranging from 1031.98 to 1049.68. The category with the highest mean Elo was Geometry (1049.68), while the lowest was Measurement and applied mathematics (1031.98). The highest variability was found in Fractions, decimals and percentages, which had a standard deviation of 61.84, suggesting a wider range of performance among students in this area. In contrast, the four types of arithmetic showed the most consistency with a standard deviation of only 22.74. This spread in scores reflects both the diversity of skill levels and the adaptive nature of the platform, which attempted to match question difficulty to student ability in real time.

Interestingly, these findings align with broader educational research. Studies have consistently shown that students struggle more with fractions and decimals due to their abstract representation and perceived complexity (Lortie-Forgues et al. [2015]). For example, the TIMSS assessment framework emphasizes that mastery of fractions and decimals, particularly converting between forms and computing, is notably challenging for middle-grade learners (Mullis, Ina V. S. and Martin, Michael O. [2017]). This external benchmark reinforces the notion that higher variability in these topics is not unique to the sample but reflects a common pattern in mathematics education.

While direct comparisons with national Danish curricula or standardized data are limited, the international trend suggests that EloQuiz results may indeed mirror broader student proficiency patterns. Measurement and applied mathematics, despite being rated lowest in Elo, may not necessarily be easier; rather, its lower mean may reflect less curriculum exposure or varied practical reasoning skills, warranting further investigation.

**User feedback and survey results** To complement quantitative data from system logs, a post-session survey was conducted among the 9 participating students. The survey aimed to assess usability, satisfaction, perceived learning value, and engagement with the app. Below is a summary of the key findings:

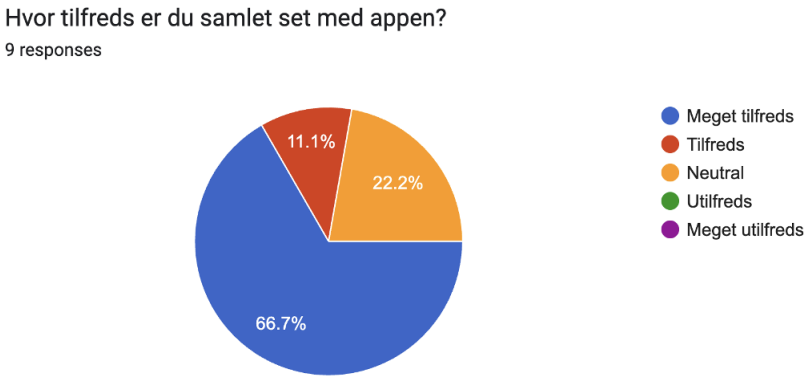


Figure 5.1: Overall, how satisfied are you with the app?

Figure 5.1 shows that 66.7% of students reported being "very satisfied", while only one student (11.1%) expressed dissatisfaction. This suggests that the platform was generally well received.

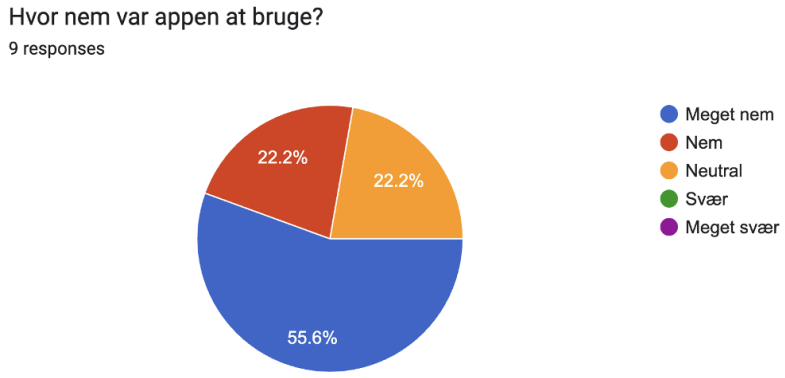


Figure 5.2: How easy was the app to use?

As shown in Figure 5.2, a majority (77.8%) found the app either "very easy" or "easy" to use. No students reported significant usability issues, indicating that the UI was intuitive even without prior instruction.

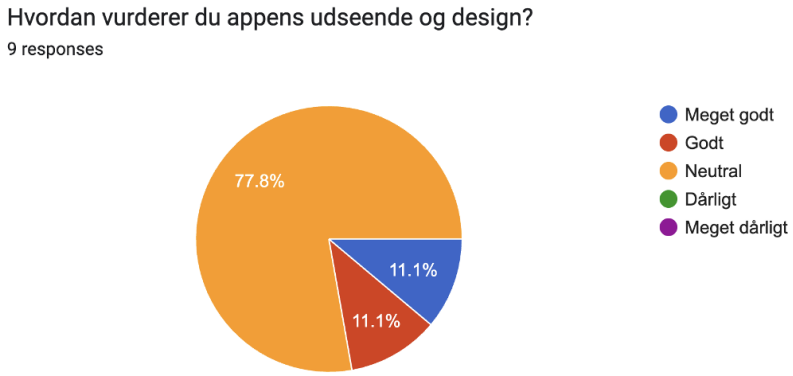


Figure 5.3: How do you rate the app's appearance and design?

Figure 5.3 reveals that while only 22.2% rated the appearance positively, the majority (77.8%) selected "neutral." This suggests the design was not distracting but perhaps not especially engaging.


Figure 5.4: Did you feel that the app helped you learn new things?

According to Figure 5.4, 33.3% felt they learned "a lot", while 44.4% indicated they learned "a little." Only 22.2% expressed doubt about the app's educational value.

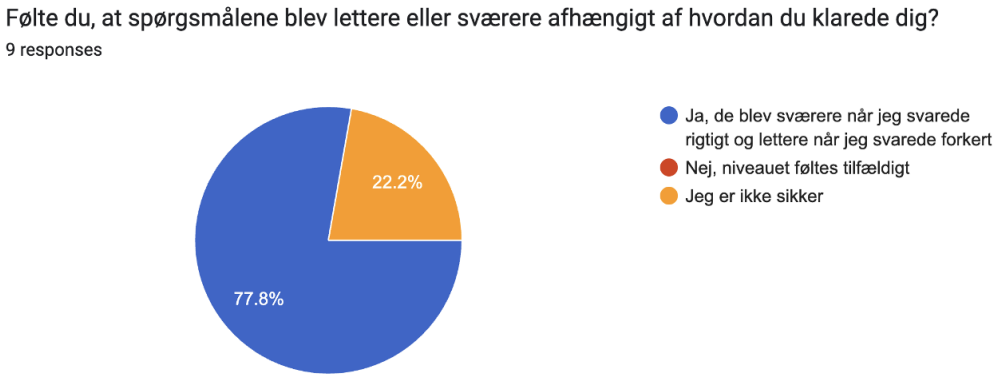


Figure 5.5: Did you feel that the questions got easier or harder depending on how you did?

In Figure 5.5, 77.8% agreed that question difficulty adjusted based on their performance, supporting that the adaptive Elo system appear to was working as intended.

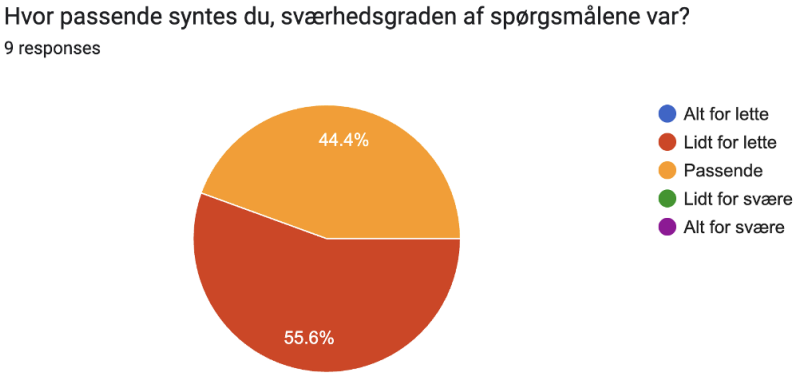


Figure 5.6: How appropriate did you find the difficulty of the questions?

Figure 5.6 suggests the difficulty curve was slightly off: 55.6% found the questions "a bit too easy", while 44.4% found them "just right."

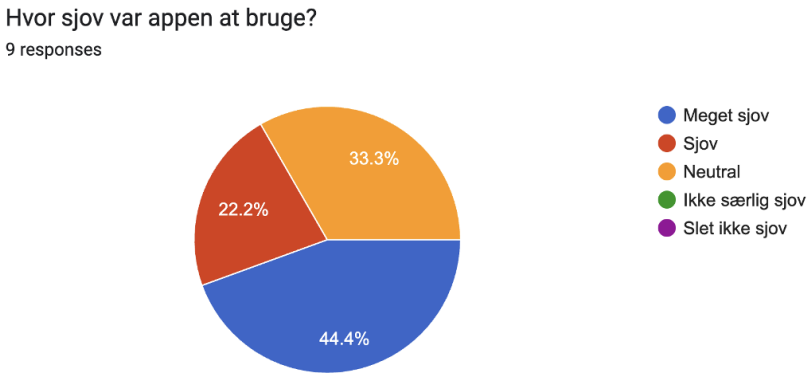


Figure 5.7: How fun was the app to use?

As shown in Figure 5.7, over 75% found the app fun or neutral in terms of engagement.

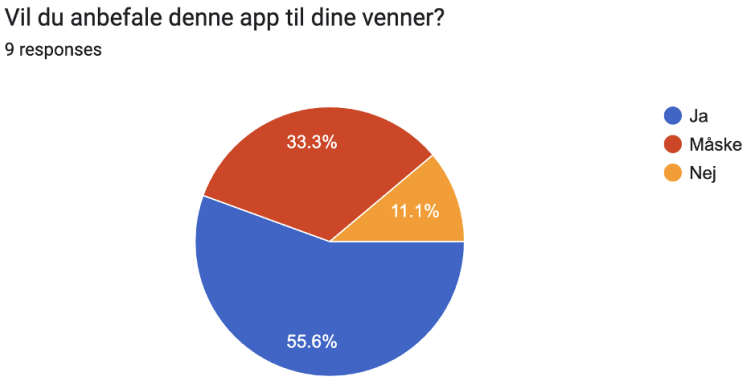


Figure 5.8: Would you recommend this app to your friends?

Finally, Figure 5.8 shows that 55.6% would recommend the app, while 33.3% said maybe.

**Backend performance observations** During the classroom testing sessions, several backend-related behaviors were observed that highlight both the strengths and current limitations of the system's technical infrastructure.

The backend, implemented using Flask and hosted on Google Cloud Run, connected to Firebase Firestore for data storage and retrieval. For most of the session, performance was stable, and response times for retrieving and submitting quiz data were acceptable within the classroom context.

However, during the second classroom session, the system reached the free daily quota limit of 50,000 Firestore document reads. This caused the backend to become unresponsive for all subsequent requests, effectively halting usage for the remaining students. Since billing was not enabled at the time, the platform was unable to scale beyond the free

quota, exposing a critical dependency on configuration and quota monitoring. This incident illustrates the importance of setting up usage alerts and ensuring billing is enabled before deployment in a live environment.

In terms of resource consumption, each quiz session appeared to generate a high number of read and write operations, particularly when loading user data, retrieving questions, updating Elo scores, and logging answers. Future optimizations should focus on reducing redundant reads, caching frequently accessed data, and batching updates where possible.

Despite these issues, the backend architecture functioned as intended under moderate load and provided a robust foundation for adaptive question delivery and real-time data updates. With further scaling considerations and performance tuning, the system is well-positioned to support larger numbers of concurrent users.

**Student behavior patterns** During the classroom testing sessions, several patterns emerged in how students interacted with the EloQuiz platform. These behaviors offer insights into user expectations, engagement levels, and usability from the perspective of younger learners.

First, many students demonstrated a clear motivation to improve their performance, often expressing excitement about increasing their streaks or advancing to more difficult questions. This suggests that the Elo-based progression system may have functioned effectively as a source of extrinsic motivation. In particular, students appeared to enjoy seeing immediate feedback and tracking their success in real time.

Navigation habits also revealed certain assumptions about interface design. Several students instinctively attempted to log in rather than sign up, indicating that the default landing page or button hierarchy may have been unintuitive. Similarly, some students began the quiz before linking to a teacher account, which implied that prominent buttons were prioritized visually over instructional flow. These behaviors reflect a tendency to interact with digital systems based on visual hierarchy and trial-and-error exploration, rather than reading instructions thoroughly, perhaps a common trait in younger users.

Interestingly, some students asked for confirmation before proceeding with actions they likely understood (e.g., “Should I click this?”). This may reflect a classroom dynamic in which students seek adult validation, rather than confusion about the system itself. A few students scrolled past important UI elements, suggesting that visual cues should be anchored persistently in view or reinforced with additional prompts.

There was also evidence of curiosity-driven exploration. Some students clicked links not directly related to the main quiz flow, such as the feedback form or teacher linking option, purely to see what would happen. One student even asked about the backend logic behind question difficulty, an indication that some learners were actively engaging with the concept of adaptivity, not just the surface-level task.

Taken together, these behaviors highlight the need for robust onboarding flows and clear visual structure when designing adaptive learning tools for younger audiences who are both eager and impatient users.

## 5.4 Observations and feedback

During the classroom sessions, both oral and written feedback were collected from the students. The feedback can be divided into several categories, highlighting bugs, usability issues, and suggestions for improvement.

**Bugs** A few students reported seeing the same question multiple times during their session. This was unexpected, as the initial database was manually cleaned for duplicates, and the code explicitly prevents repeated questions for the same user. This issue could not be reproduced during testing and remains unexplained, warranting further investigation.

Another student reported that their correct answer streak unexpectedly reset. This behavior was also tested and could not be replicated, but it suggests that edge cases in the streak-tracking logic may require closer examination.

**Design misunderstandings** Several students attempted to register via the login page, which is currently the default landing page. Although a "Sign Up" button is present, its placement and size may have contributed to the confusion. A potential solution would be to set the sign-up page as the default entry point or increase the visual prominence of the button.

Some students began the quiz before linking with a teacher, likely due to the “Start Quiz” button being placed above the “Link with Teacher” button on the home page. This could be resolved by prompting users to link with a teacher immediately after registration.

One student, after scrolling down, was unable to locate the frontpage navigation button, which was positioned in the header. This suggests the need for a persistent navigation option or a visible back button throughout the interface.

Another issue occurred when a student opened the feedback form in a new tab and struggled to return to the quiz. Adding a “Return to Quiz” link on the confirmation screen could address this confusion.

Despite these minor issues, the overall feedback on the interface was positive. Students generally found the design simple and intuitive.

**Backend misunderstandings** Some students entered invalid email addresses (e.g., missing ".dk") or too-short passwords, which triggered Firebase authentication errors such as:

`Firebase: Error (auth/invalid-email)` and

`Firebase: Password should be at least 6 characters (auth/weak-password).`

These are standard backend messages, but future versions of the interface could display more user-friendly validation prompts.

**Content and syntax confusion** Some students were unfamiliar with mathematical symbols or terminology. For instance, the asterisk $*$ for multiplication was not recognized by all students, and terms like "base of a triangle" and "hypotenuse" were not universally understood. Given the age group, this is expected. Future iterations of EloQuiz could include optional hints or explainations to clarify terms when needed.

**Suggested tweaks** One student suggested reducing the size of the difficulty jumps between questions. This may indicate that the adaptive algorithm could be made more gradual in future iterations.

**Feature requests** Students provided a wide range of suggestions for future features, including:

- A leaderboard to track the highest streaks among students.
- Timed questions to add pressure and excitement.
- A computed average Elo score in addition to topic-specific Elos.
- More frequent questions from topics where the student performs poorly.
- The option to select a preferred topic or category.
- Revisiting previously answered incorrect questions for reinforcement.
- Choosing an initial Elo manually to set baseline difficulty.
- A multiplayer "quiz battle" mode to compete against classmates.

**Student curiosity** Several students asked broader questions such as "How many questions are there?", "Who made these questions?", and "I know Elo from chess – how does it work here?". This shows engagement with the underlying system and a general interest in how the platform functions.

**Positive feedback** The system received a number of compliments. Students appreciated the clean design, and several commented that the platform worked well. One student even asked if they could use the site at home, indicating a high level of interest.

**Notably absent feedback** No students reported that a question was outright incorrect. This could either be because the question set was successfully filtered for errors prior to testing, or because students may have been reluctant to challenge the content, especially in a classroom setting.

**Summary of key findings** The evaluation of EloQuiz yielded several notable findings across both quantitative performance data and qualitative feedback:

- **Adaptive performance:** Students exhibited an average Elo rating of approximately 1045 across five math categories, with variation indicating differentiated skill levels. Most students showed gradual increases in Elo, suggesting the system adapted effectively to individual performance.
- **System usage:** A total of 230 questions were answered across 9 students, averaging approximately 25.5 questions per participant, despite limited testing time and technical disruptions.
- **User satisfaction:** Survey responses indicated that 66.7% of students were "very satisfied" with the app, and 77.8% reported that it was easy or very easy to use.
- **Engagement and motivation:** Students responded positively to real-time feedback and performance tracking. Several expressed interest in using the app outside of class and asked about system mechanics.
- **Perceived adaptivity:** 77.8% of students noticed that question difficulty changed based on their performance, aligning with the system's design goals.
- **Technical and design issues:** Limitations in backend scalability and interface layout led to some usability challenges and confusion during account setup and navigation.
- **Suggestions for improvement:** Students proposed features such as multiplayer quizzes, a leaderboard, and the ability to revisit incorrect answers highlighting a desire for greater interactivity and personalization.

Together, these results suggest that EloQuiz successfully delivered an adaptive learning experience that was intuitive, engaging, and pedagogically promising, even within the constraints of limited testing conditions.

# 6 Discussion

## 6.1 Key learnings from user testing

The classroom testing of EloQuiz provided valuable insights into both the technical robustness of the system and the user experience from the students' perspective. Several key learnings emerged from the sessions, which will inform future iterations of the platform.

**1. Infrastructure must be prepared for real usage** Despite extensive development and internal testing, the live environment exposed infrastructure vulnerabilities. The accidental exhaustion of Firestore's daily read quota during the second session highlighted the importance of enabling billing and monitoring usage limits proactively. Similarly, the initial firewall block underscored the need to test access in the target deployment environment well in advance.

**2. Young users require clearer guidance** Design assumptions that seemed intuitive during development did not always align with student behavior. For instance, several students attempted to sign up via the login page or skipped the "Link with Teacher" step. These misunderstandings point to the importance of optimizing onboarding flows, emphasizing key actions, and guiding users step-by-step with clear prompts.

**3. Visual hierarchy affects user flow** Button placement and page structure strongly influenced student behavior. The prominence of the "Start Quiz" button over the "Link with Teacher" option led to incorrect usage sequences. Similarly, navigation elements placed only in the header were not always visible to students who had scrolled down. Future iterations should incorporate persistent navigation elements and more deliberate visual hierarchy.

**4. Children offer unfiltered and valuable feedback** Students were highly responsive and vocal about their experiences, providing a range of suggestions for additional features. Ideas such as leaderboards, topic selection, reattempting incorrect questions, and competitive quiz battles indicate a desire for increased personalization and gamifi-

cation. This feedback demonstrates the value of involving end users, especially children, early in the development process.

**5. Age-appropriate content and language are critical** Some students were confused by symbols or terminology that might be considered basic at higher education levels. Terms like "hypotenuse" or the multiplication asterisk (∗) were not universally understood. This illustrates the importance of either simplifying language or providing embedded hints, especially in tools aimed at younger learners.

**6. Testing in the real world is indispensable** While local development and testing can identify many issues, real-world testing revealed edge cases and usage patterns that would otherwise go unnoticed. These findings reinforce the value of iterative development cycles that include live user testing with actual target audiences.

## 6.2 Limitations

While the development and testing of EloQuiz yielded valuable insights, several limitations must be acknowledged that may have affected the scope and generalizability of the results.

First, the classroom testing ended up being conducted with only nine 5th grade students due to technical and logistical issues. The small sample size and narrow age range limit the ability to generalize findings across different grade levels, subjects, or school environments.

Second, the first part of the research question asked whether an Elo rating system can accurately predict a student's proficiency level. While the Elo system is designed to converge to an accurate score over time, this thesis did not include a comparison between Elo ratings and independent benchmarks, such as student grades. Such a comparison would have provided stronger evidence for the accuracy of the system's proficiency predictions.

Third the platform was only tested with math content. Although the adaptive system and Elo-based logic are designed to be generalizable to other subjects, this assumption has not been empirically tested. The effectiveness of Elo-based question selection may vary significantly in non-quantitative domains such as language arts or social studies.

Fourth, because the questions were generated using OpenAI's language models, some variation in question phrasing and difficulty may have occurred, despite internal checks. While the AI-generated questions were filtered for quality, the evaluation of content appropriateness was not systematically measured during the sessions.

Additionally, teacher feedback was not formally collected during the testing phase, which limits the assessment of EloQuiz from a pedagogical or classroom management

perspective. Future iterations would benefit from more structured input from educators, particularly regarding integration into existing curricula.

Finally, due to time constraints and the focus on proof-of-concept development, several advanced features such as long-term performance tracking, gamified feedback, and detailed analytics for teachers were left unimplemented or only partially functional. These limitations reflect the MVP (minimum viable product) nature of this version of EloQuiz and point to areas for future development and research.

## 6.3 Challenges in modern education

While developing EloQuiz, it became evident that the system does not operate in a vacuum. It is shaped by and responds to several systemic issues in contemporary education. This section outlines key challenges that informed the design and relevance of the platform.

**Standardization and test-centric learning** Many educational systems prioritize standardized testing, which incentivizes memorization over deep understanding. This "teaching to the test" approach can limit critical thinking, creativity, and real-world application. Adaptive platforms like EloQuiz aim to offer a more dynamic alternative by responding to individual student performance in real-time.

**Educational inequality** Access to quality learning resources is unevenly distributed across socio-economic groups. Factors such as internet access, school funding, and teacher availability create disparities in educational outcomes. While EloQuiz alone cannot close this gap, its low infrastructure requirements and flexible accessibility offer some mitigation.

**Lack of personalization** Traditional classrooms often follow a one-size-fits-all model that may not accommodate varying levels of ability, learning styles, or pacing needs. This leads to disengagement among both advanced and struggling students. EloQuiz's use of adaptive difficulty based on the Elo rating system directly addresses this by tailoring the challenge level to individual users.

**Teacher burnout and administrative load** Educators face increasing demands, including content delivery, behavioral management, and administrative documentation. Burnout is a growing concern, especially in under-resourced schools. By automating aspects of formative assessment, EloQuiz has the potential to reduce workload and free up time for more meaningful interaction between teachers and students.

**Slow technological adoption** Although educational technology has advanced rapidly, its adoption within school systems is often hindered by bureaucracy, lack of training, and resistance to change. EloQuiz was designed to be lightweight, intuitive, and minimally invasive to lower the barrier to integration.

**Curriculum relevance and skill gaps** There is a growing disconnect between what students are taught and the skills they need in real life. Topics like financial literacy, digital citizenship, and critical thinking are often underrepresented. While EloQuiz focuses on math in its current form, the underlying framework could be expanded to support more diverse, relevant content areas in the future.

## 6.4 Implications

The development and classroom testing of EloQuiz suggest several broader implications for educational technology, particularly in the context of adaptive learning, student engagement, and the use of AI-generated content in primary education.

First, the use of an Elo-based system to match students with appropriately difficult questions appears promising. Even within a short testing period, the system encouraged engagement by offering a personalized level of challenge. This supports the idea that adaptive mechanisms borrowed from gaming and competitive systems can be effectively repurposed to maintain motivation and prevent frustration or boredom in learning environments.

Second, the observed student behavior and feedback suggest that younger students are not only capable of using digital learning platforms independently, but are also highly responsive to game-like feedback systems such as streaks, score progression, and difficulty scaling. This reinforces the value of integrating lightweight gamification into learning tools, particularly in age groups where attention spans are short and intrinsic motivation may vary.

The successful generation and deployment of questions via a language model (OpenAI) also points to the practical utility of AI-generated content in classroom settings. While human oversight remains essential to ensure accuracy and appropriateness, the ability to rapidly produce diverse, level-specific questions could significantly reduce the content creation burden for educators, especially in under-resourced schools.

Finally, the prototype's deployment in real classrooms, despite technical limitations, demonstrates that even relatively simple, independently developed educational tools can function in real-world school settings and generate useful feedback. With further development, teacher-facing dashboards, and integration into existing learning platforms, systems like EloQuiz could complement traditional instruction and support differentiated learning at scale.

## 6.5 Future work

While EloQuiz currently functions as a minimum viable product focused on adaptive math quizzes for middle school students, the results of this project point to numerous directions for future development, testing, and research.

First, additional large-scale testing is needed across a broader range of age groups, subjects, and school settings. The current testing environment was limited in scope, and future studies could examine how students at different educational levels respond to adaptive learning systems and whether similar engagement patterns hold in other domains such as science, language arts, or history.

Second, further refinement of the adaptive algorithm could enhance personalization. One promising direction is improving the granularity of difficulty scaling by incorporating more data points, such as response time and question type, into the Elo updates. Additionally, implementing mechanisms for revisiting previously incorrect questions or targeting weak topics more aggressively could improve learning retention and provide a more coherent progression path for students.

Third, the current teacher interface is minimal and could be expanded into a full-featured dashboard. Features such as real-time class performance tracking, topic heatmaps, and assignment creation would allow educators to better integrate the tool into classroom instruction and monitor student progress over time.

In terms of content generation, future work could focus on enhancing the quality and variety of AI-generated questions by fine-tuning prompts, incorporating curriculum alignment, and adding multimedia elements such as diagrams or interactive components. Developing a hybrid content pipeline, combining AI generation with teacher-created or curated content, could also ensure both scalability and pedagogical quality.

From a technical perspective, improved error handling, backend optimization (e.g., reduced Firestore reads), and scalability planning are necessary steps for transitioning from a prototype to a production-ready system. Supporting multiple languages and accessibility features would also increase the platform's inclusivity and usability.

To reduce the likelihood of incorrect or poorly phrased AI-generated questions, a flagging system could be implemented to allow students and teachers to report problematic content directly. This feedback could be stored and reviewed to improve content quality over time. Additionally, an automated validation layer could be introduced, where a secondary API prompt reviews and verifies the output of the original prompt before the question is presented to users. This multi-step generation pipeline could reduce the need for manual review and increase the reliability of the question database.

Finally, future iterations could explore social and motivational features such as student leaderboards, classroom competitions, and progress badges. While gamification must be balanced with educational intent, these features were frequently requested by students

and could play a role in sustaining long-term engagement.

In summary, EloQuiz provides a strong foundation for continued development as an adaptive educational platform, and future work could significantly expand both its technical capabilities and its pedagogical impact.

# 7 Conclusion

This project set out to explore whether an Elo rating system, originally developed for competitive games such as chess, could be effectively applied to the context of educational assessment and question delivery. Specifically, the aim was to investigate whether such a system could predict student proficiency levels and deliver questions that are appropriately challenging in real time.

Through the design and implementation of EloQuiz, a functional adaptive learning platform was developed, integrating question difficulty scaling with student performance tracking via topic-specific Elo scores. Classroom testing with 5th-grade students, while limited in scale, provided promising evidence that the system could successfully adjust question difficulty based on prior performance, and that students responded positively to this adaptive approach. Students appeared engaged, were curious about their progression, and displayed motivation to improve their streaks and scores, indicating that the Elo mechanism had a meaningful effect on perceived challenge and achievement.

However, while the Elo system provided a lightweight and flexible means of estimating proficiency, several limitations were identified. The experience gathered from the test supports that the system especially when rolled out on a larger populace will be able to predict (or rather assess) the student's level of proficiency on the basis of the answers provided during the tests by the student and by other tested participants. The system is designed so that it is to be expected that the student's Elo score will converge and the difficulty of the questions posed to the student will converge against a level of difficulty reflecting the student's actual proficiency. The system does not take into account factors such as concept mastery, response quality, or learning over time. It also assumes a level of independence between questions and treats performance as a binary correct/incorrect outcome, which may oversimplify the nuances of learning.

Despite these constraints, the results suggest that an Elo-based model can serve as a useful approximation of student ability in a digital quiz environment, particularly when combined with topic segmentation and a carefully tuned difficulty scale. It offers a viable and computationally efficient alternative to more complex adaptive systems, especially in low-resource or exploratory settings.

In conclusion, while Elo ratings may not capture the full complexity of student learning, they can provide a practical and effective framework for delivering optimally chal-

lenging questions in real-time educational systems. With further refinement, integration of additional data, and expansion into broader subject areas, systems like EloQuiz have the potential to contribute meaningfully to the development of adaptive learning tools that support personalized, scalable, and engaging education.

# Bibliography


Arpad Elo. *The Rating of Chessplayers, Past and Present*. Arco Pub, 1978.

5Rights Foundation. Ai systems that exploit the vulnerabilities of children are now illegal in the eu, 2024. URL `https://5rightsfoundation.com/ai-systems-that-exploit-the-vulnerabilities-of-children-are-now-illegal-in-the-eu/` Accessed May 2025.

Mark E Glickman. A comprehensive guide to the glicko rating system. *Boston University Department of Mathematics*, 1995.

Hugues Lortie-Forgues, Jing Tian, and Robert S. Siegler. Why is learning fraction and decimal arithmetic so difficult? *Developmental Review*, 2015.

Mullis, Ina V. S. and Martin, Michael O. *TIMSS 2019 Assessment Frameworks*. International Association for the Evaluation of Educational Achievement (IEA), 2017. URL `https://timssandpirls.bc.edu/timss2019/frameworks/`.

European Parliamentary Research Service. The eu artificial intelligence act, 2025. URL `https://www.europarl.europa.eu/RegData/etudes/ATAG/2025/769494/EPRS_ATA(2025)769494_EN.pdf`. Accessed May 2025.

SoBigData. Children's vulnerability and the eu ai act, 2024. URL `https://www.sobigdata.eu/blog/childrens-vulnerability-eu-ai-act`. Accessed May 2025.

Wayne Xin Zhao, Zhirui Shao, Jingyuan Li, Hongyi Wang, Haifeng Wang, and Ji-Rong Wen. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

# A Appendix A: Use of large language models (LLMs) in the project and thesis

Large Language Models (LLMs), specifically OpenAI's GPT-based tools, were used in various supportive capacities during both the development of EloQuiz and the writing of this thesis. Their role was primarily as an assistant, helping to increase efficiency, provide technical clarity, and act as a collaborative brainstorming tool, rather than as an autonomous content generator.

In the Development of EloQuiz LLMs were used to:

- Help refine API prompts for generating math questions based on topic and difficulty
- Suggest Python and JavaScript code snippets, especially for handling Firestore operations, user session logic, and error handling
- Troubleshoot bugs or review logic in backend functions like Elo calculation and question selection
- Offer advice on optimization and architecture (e.g., caching strategies, CORS configuration)

These uses were always followed by human oversight, testing, and adjustment. The final implementation reflects a combination of AI-suggested ideas and manual problem-solving.

During the thesis process, LLMs were used to:

- Rephrase sections for clarity and improve academic tone
- Provide structural suggestions (e.g., how to organize chapters like Results and Discussion)
- Give feedback on drafts, such as identifying repetition or offering more precise terminology
- Generate LaTeX snippets and help with formatting (e.g., tables, figures, citations)

Importantly, all content was written with human intent and review. The LLM was

used similarly to how one might use a writing assistant or editor, to enhance fluency, correct phrasing, or confirm understanding, rather than to replace the writing process itself.

The use of LLMs in this project reflects a broader trend in both education and software development, where such tools are becoming integral to workflows. In this case, they supported the learning process, encouraged experimentation, and allowed more time to focus on creative and critical aspects of the project.

Transparency in the use of these tools is important, especially in academic work, and every effort has been made to ensure that all contributions from LLMs were appropriately reviewed, edited, and integrated by the author.

# B Appendix B: Screenshots

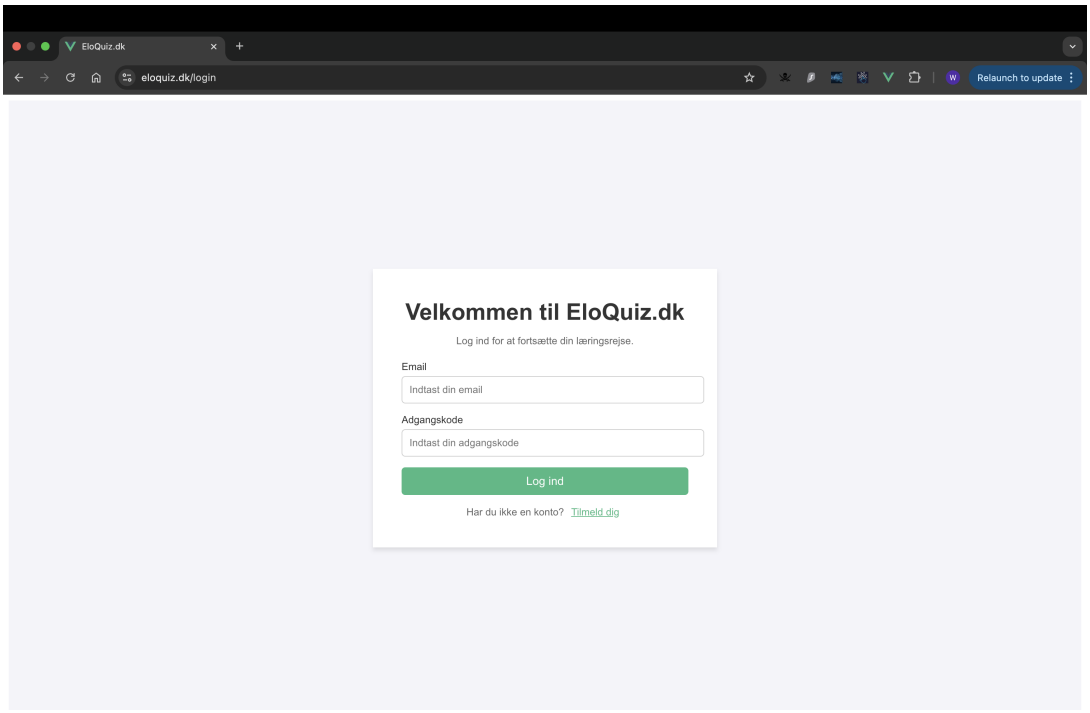


Figure B.1: UserLogin

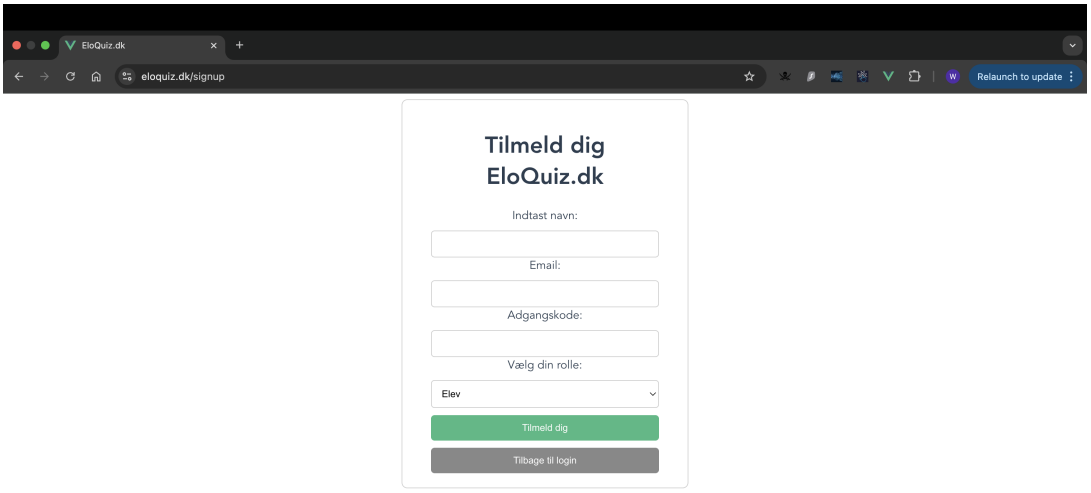


Figure B.2: UserSignup

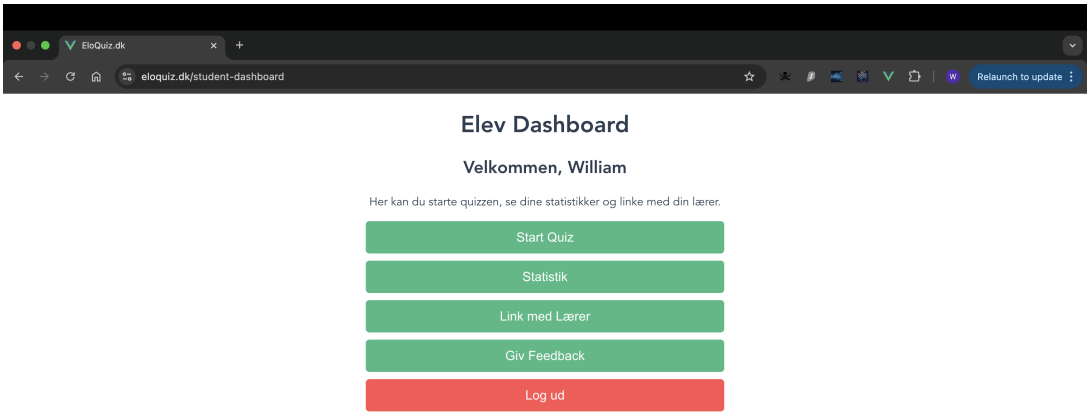


Figure B.3: StudentDashboard

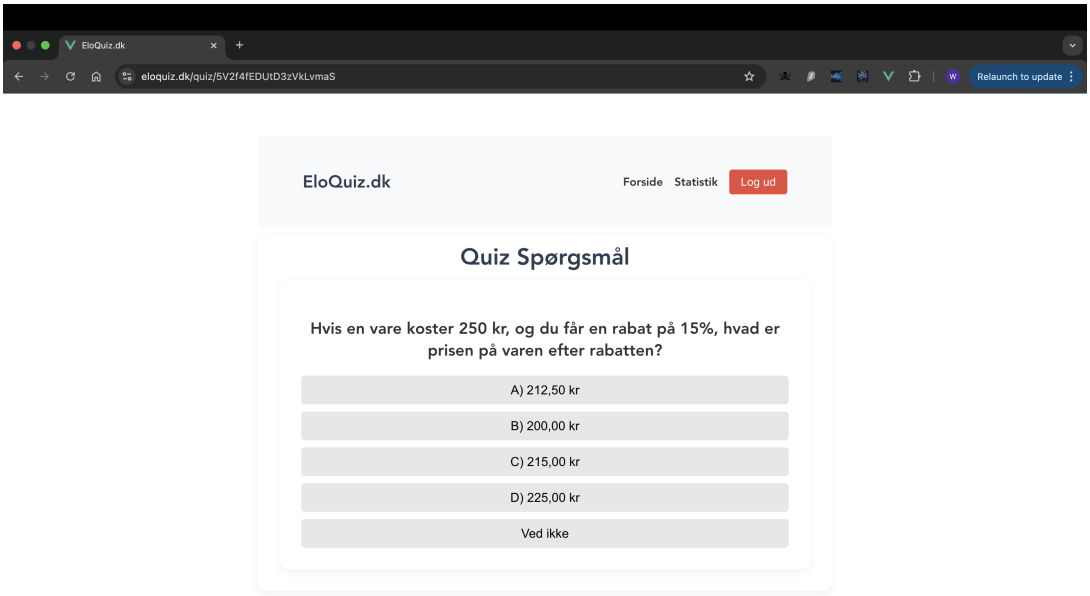


Figure B.4: QuizScreen

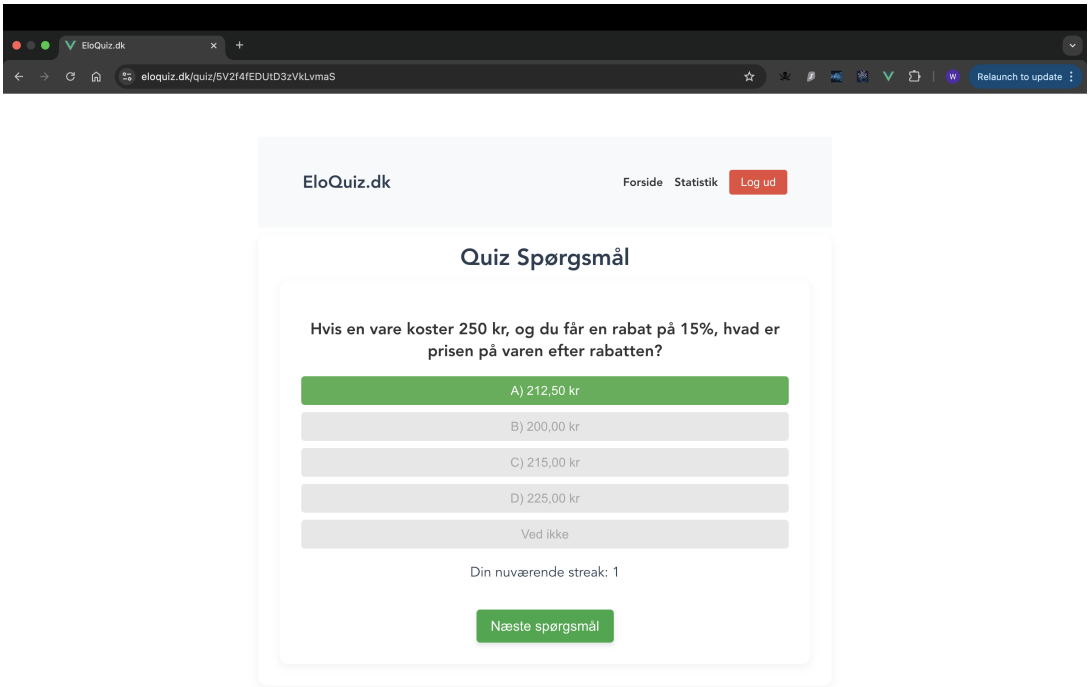


Figure B.5: QuizScreen after answering question

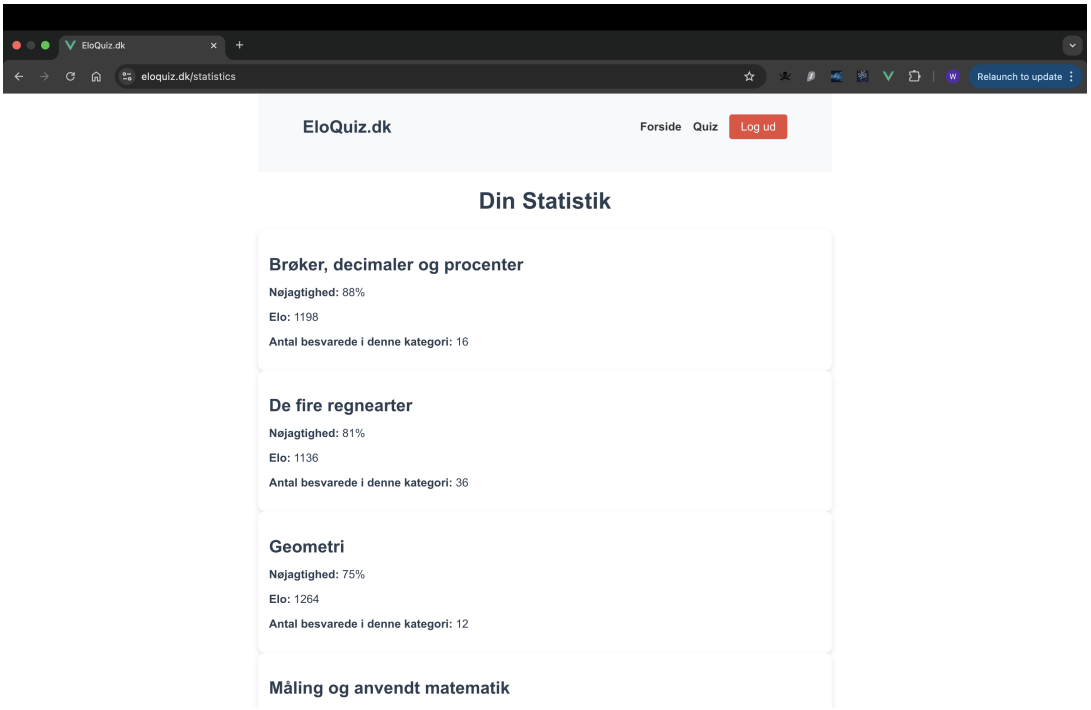


Figure B.6: StatisticsPage

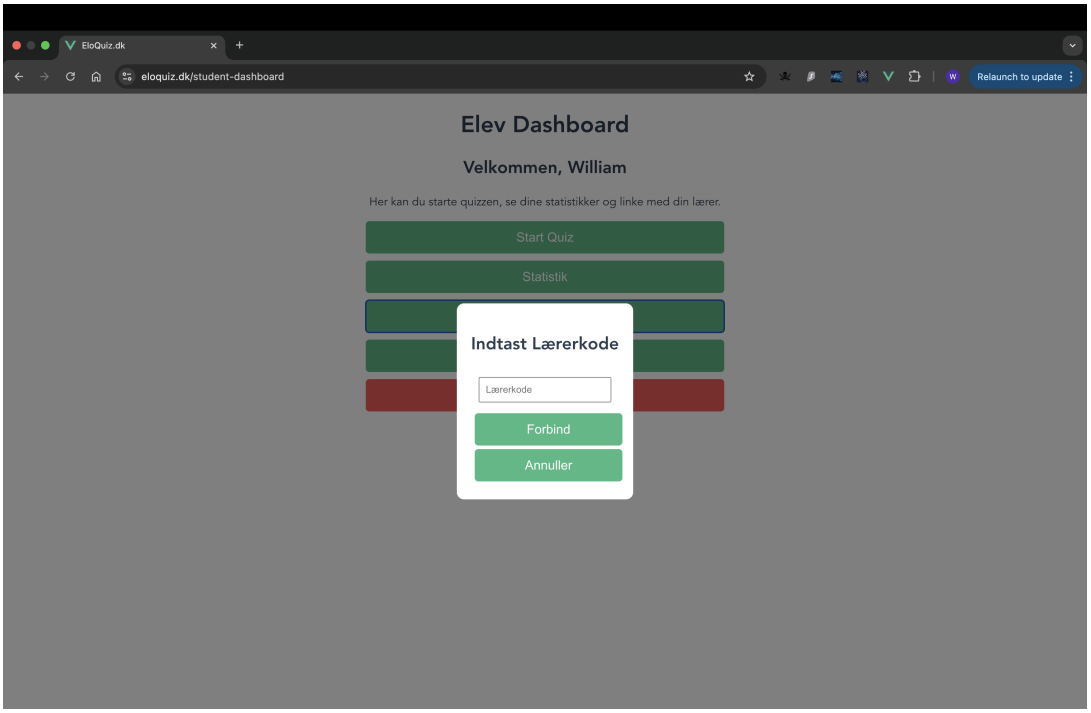


Figure B.7: StudentLink

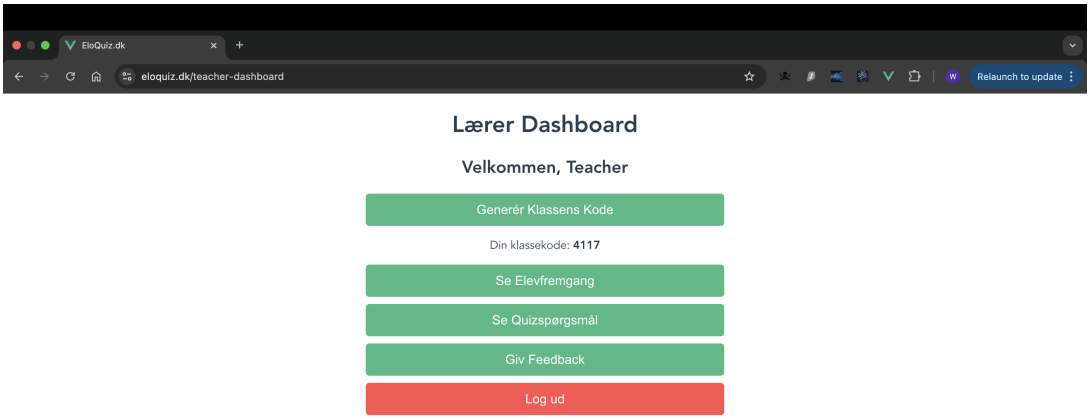


Figure B.8: TeacherDashboard

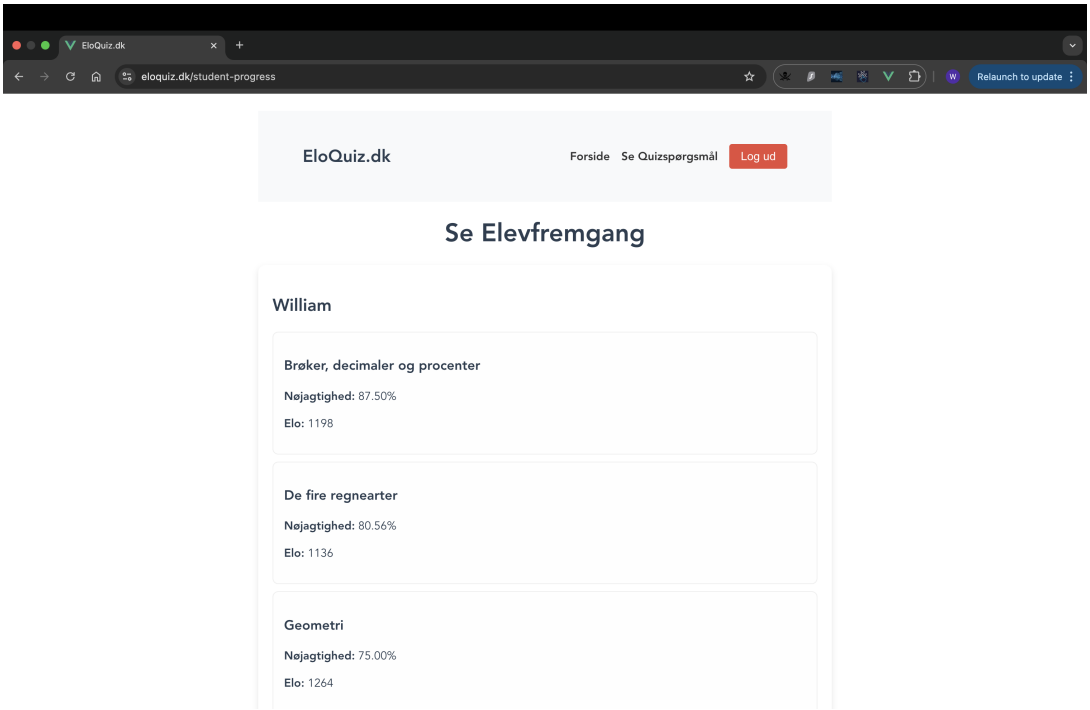


Figure B.9: StudentProgress

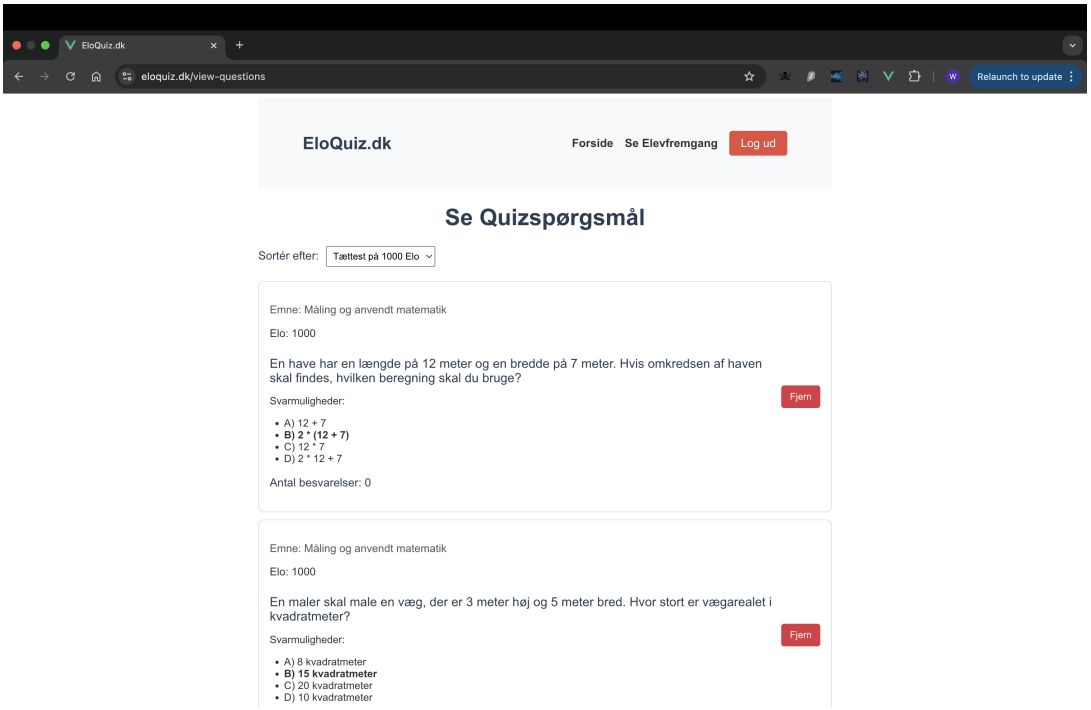


Figure B.10: ViewQuestions

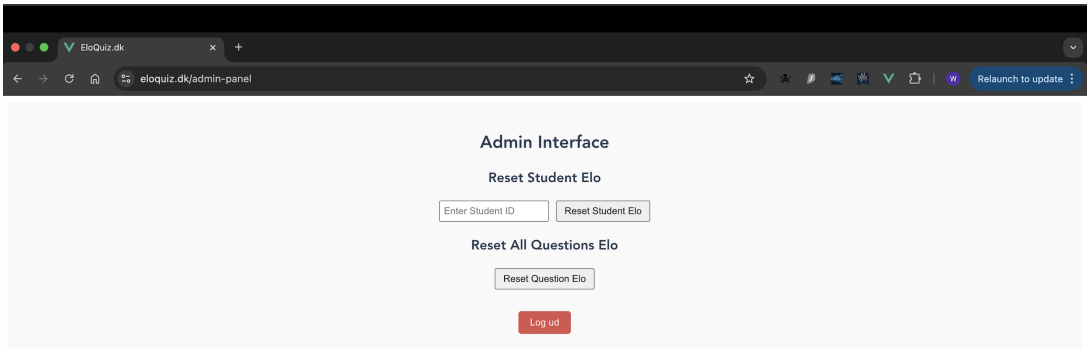


Figure B.11: AdminPanel

# C Appendix C: Remaining views from implementation chapter

**AdminPanel.vue** The AdminPanel.vue view provides a dedicated administrative interface within the application. This view is only accessible to users authenticated as administrators and serves as a centralized dashboard for managing key elements of the system related to the Elo-based adaptation engine.

The interface allows administrators to perform two main functions:

· Reset Student Elo: This feature enables the admin to reset the Elo rating of an individual student. The student ID is entered into a text input field, and upon clicking the associated button, a POST request is sent to the backend endpoint /api/reset_student_elo. This is useful in cases where a student's rating needs to be recalibrated due to anomalies or testing resets.

· Reset All Question Elos: This option resets the Elo ratings for all quiz questions in the database. It sends a POST request to /api/reset_question_elo and includes a confirmation prompt to prevent accidental misuse. This functionality can be helpful for system-wide resets, such as when conducting a new pilot test or after major content revisions.

In addition, the component includes a logout button that redirects the user to the login screen using Vue Router. Success and error messages are displayed conditionally to inform the admin of the result of their actions.

Overall, AdminPanel.vue provides essential administrative controls for maintaining and testing the adaptive learning system's Elo-based logic, ensuring data integrity and offering flexibility for educational interventions.

**RedirectHandler.vue** The RedirectHandler.vue view is responsible for determining a logged-in user's role and redirecting them to the appropriate section of the application. It is implemented as a minimal view that automatically executes logic on page load, displaying only a brief "Redirecting. . . " message during processing.

Upon mounting, the component checks the currently authenticated user via Firebase Authentication. If a user is detected, it retrieves the corresponding document from the users collection in Firestore to determine their role. Based on the retrieved role, the user

is redirected to one of approiate dashboard, either student, teacher or admin.

If no authenticated user is found, or if the role cannot be retrieved, the user is redirected to the login page.

This view plays a key role in handling role-based routing and access control immediately after login or authentication, ensuring that users are taken directly to their appropriate dashboard without additional manual input.

**StudentLink.vue** The StudentLink.vue component allows students to associate their account with a specific teacher using a unique class code. This functionality is essential for linking student activity and performance data with a corresponding teacher account, enabling classroom management and targeted oversight.

Upon rendering, the component displays a simple interface with a text input field and a button labeled "Tilslut." The student enters a class code provided by their teacher and clicks the button to initiate the linking process.

Internally, the following steps occur:

1. Authentication Check: The component verifies that the student is authenticated via Firebase Authentication. If not, an alert prompts the user to log in.

2. Database Query: The component accesses the Firebase Realtime Database and retrieves all teacher entries stored under the teachers node. It searches for a match between the entered class code and any existing teacher's class_code property.

3. Validation: If no matching teacher is found, the student is notified that the code is invalid. If a match exists, the linking process continues.

4. Bidirectional Linking:

- A new entry is created under students/studentId, storing the teacher_id and timestamp.
- Simultaneously, the student is added under teachers/teacherId/students/studentId with a timestamp.

5. Navigation: Upon successful linking, the student is redirected to the main student dashboard (/student/dashboard).

This component ensures that each student can only join an existing teacher's group via a valid code, preserving controlled access and enabling personalized oversight within the system. It also lays the groundwork for tracking student data in a teacher-centric structure, facilitating performance analysis and future teacher-facing functionality.

**StudentProgress.vue** The StudentProgress.vue component provides teachers with a detailed overview of their students' performance within the platform. It serves as a progress monitoring interface and is only accessible to authenticated teacher accounts.

Upon mounting, the component retrieves the teacher's user ID via Firebase Authentication. It then makes a request to the backend endpoint /api/get_students, which

returns a list of students associated with that teacher. Each student object includes statistical data categorized by topic.

For each student, the interface displays:

- The student's name (or user ID if unavailable),
- A breakdown of performance per topic, including:
- Accuracy (as a percentage),
- Elo rating (rounded for clarity).

The statistics are presented in neatly formatted "cards" for readability, with visual emphasis on individual topics to facilitate quick evaluation.

Additionally, the component includes a feature for removing students. Clicking the "Fjern elev" button triggers a confirmation dialog and, upon confirmation, sends a DELETE request to /api/remove_student. If the operation succeeds, the component refreshes the student list to reflect the update.

Overall, StudentProgress.vue acts as an essential administrative tool, enabling teachers to track student growth, identify potential issues, and maintain an up-to-date roster.

**NavBar.vue** The NavBar.vue is component rather than a view and provides the top navigation bar used across the application. It is a dynamic and reusable component that adjusts its content based on the current user role and page context, ensuring that navigation remains intuitive and relevant to the user.

# D Appendix D: Code repository and system overview

The source code for EloQuiz is publicly available at:

`https://github.com/WPeytz/EloQuiz`

The platform is an adaptive, cloud-deployed learning tool developed as part of this thesis to explore the use of Elo ratings and AI-generated content in middle school mathematics education. The repository contains all components required to run and deploy the system.

## Features and functionality

- **Adaptive Difficulty:** Quiz questions adjust in real time according to each student's Elo rating.
- **AI-Generated Content:** Questions are generated on demand using OpenAI's GPT models for multiple math topics and difficulty levels.
- **Elo-Based Scoring:** Both students and individual questions are rated using an Elo formula inspired by competitive ranking systems.
- **Role-Based Dashboards:** Interfaces are tailored for students, teachers, and administrators.
- **Performance Analytics:** Real-time tracking of student progress and topic proficiency.
- **Cloud Deployment:** Hosted on Google Cloud Run using Docker containers.

## System architecture

The project is divided into the following major components:

- `frontend/` – A Vue.js-based single-page application (SPA) that handles user interaction and renders role-specific dashboards.
- `backend/` – A Flask API that handles user management, Elo updates, question selection, and integration with OpenAI for content generation.
- `.env.example` – A template for configuring API keys and cloud credentials.
- `deploy.sh` – A Bash script that builds and deploys the project to Google Cloud Platform using Docker.

## Technology stack

EloQuiz integrates the following tools and services:

- **Frontend:** Vue.js
- **Backend:** Python 3.10 with Flask
- **Database and Authentication:** Firebase Firestore
- **Content Generation:** OpenAI GPT-4
- **Deployment:** Google Cloud Run with Docker

## Environment and setup

To run the platform locally, the following components are required:

- Node.js and npm
- Python 3.10+
- Firebase project and service account credentials
- OpenAI API key

Instructions for local setup and deployment are provided in the README on the repository.

## License