ELO Learning

REQUIREMENTS DOCUMENT

ZERO DAY

University of Pretoria

Name	Student number
RM (Rene) Brancon	u22556771
NF (Nigel) Mofati	u22528084
TM (Tukelo) Mokwena	u22536800
S (Saskia) Steyn	u17267162
NG (Ntokozo) Tonga	u22506773

Team contact:
ZeroDay0D4y@gmail.com

Contents

1 Introduction..... 3
1.1 Purpose of the Document.....3
1.2 Scope of the System..... 3
1.3 Intended Audience..... 3
2. User Stories..... 4
1. Students (Primary Users)..... 4
2. System..... 5
Adaptive Testing & ELO Calculation.....5
Submission Evaluation & Feedback.....5
Tracking & Analytics.....6
Leaderboard and Abuse Prevention.....6
Notifications & Engagement.....6
3. Admin (Developers and Stakeholders).....6
Content & Question Management.....6
User Account Management.....6
Data Protection & Compliance..... 7
Analytics & Trends.....7
3. System Requirements..... 8
FR1: User Registration and Profile Creation..... 8
FR2: Secure Login and Authentication.....8
FR3: Baseline Testing & Adaptive Questioning.....8
FR4: Game Modes and Math Practice..... 8
FR5: Math Keyboard Input..... 8
FR6: Feedback and Memorandum.....9
FR8: ELO & Performance Analytics..... 9
FR9: Push Notifications.....9
FR10: User Accessibility..... 9
FR11: Admin – Content & Question Management..... 9
FR12: Admin – User Management.....10
FR14: Admin – Reporting and Analytics..... 10
4. Use Cases..... 11
5. Architectural Diagram.....16
5. Domain Model.....17
6. Math Sections.....18
Grade 8:.....18
Grade 9:.....18
Grade 10:.....18
Grade 11 + 12:.....19
Technology Choices..... 19
Frontend Development: React.js and Next.js (PWA)..... 19
Backend Development: Express.js.....19
Database Strategy: PostgreSQL + InfluxDB..... 20

Real-Time Communication: NestJS WebSocket Gateway.....20
Authentication: OAuth 2.0 + JWT.....20
Services Contracts.....20
Architecture Requirements.....41

7. WOW FACTORS…….50

1 Introduction

1.1 Purpose of the Document

The purpose of this document is to define the user stories, use cases, functional and non-functional requirements for the ELO Learning platform. It outlines the objectives of the system, describes the features and interactions expected from users, and sets the foundation for system design and development. This document serves as a point of reference for our development team, project stakeholders, and any future maintenance efforts.

1.2 Scope of the System

ELO Learning is a gamified math app designed to help students from Grade 8 to first-year university level improve their mathematical proficiency. Using an ELO-based rating system inspired by competitive games, the platform matches students with questions that reflect their skill level.

Students begin by filling in their profile and completing a baseline test, which dynamically adjusts in difficulty using a decision tree structure. Based on their performance, the system assigns them an initial ELO score. The platform includes features such as secure user authentication, gamification elements (badges, leaderboards), and personalised analytics to promote engagement and continuous learning.

1.3 Intended Audience

This document is intended for:

- **Developers and Designers:** To understand the system functionality and implement the design accordingly.
- **Project Stakeholders (Proking Solutions):** To ensure the platform meets business and user goals.
- **COS 301 Supervisors and Evaluators:** To assess the completeness and feasibility of the project.
- **Testers:** To develop test plans and verify the system against requirements.
- **Future Maintenance Teams:** To provide a clear reference for enhancements or troubleshooting.

2. User Stories

The user stories are discussed from 2 different perspectives, that of students, and general users.

1. Students (Primary Users)

Grade 8 to first-year university students.

1. As a student, I want to sign up and create my profile by providing my personal (name, surname, username, email address) and academic details (age, grade, and confidence level) so that the platform can tailor the experience to my level.

2. As a student, I want the ELO algorithm to balance my rating based on matches I play..

3. As a student, I want to answer questions using a math keyboard so that I can input notation like fractions and exponents correctly.

4. As a student, I want to view my current ELO rating and performance stats (such as accuracy and progress over time) so that I can track my improvement.

5. As a student, I want to choose between different game modes– Ranked Matches, Practice Rounds, and Single player Rounds– so that I can learn in a way that suits my goal (be it to progress or speed).

6. As a student, I want to solve math problems in Ranked Matches based on my ELO so that I can improve my rating and feel challenged.

7. As a student, I want to enter Practice mode with questions at my current level so that I can improve at my own pace.

8. As a student, I want to attempt Single player matches where I solve as many questions as possible under a time limit so that I can sharpen my speed and accuracy.

9. As a student, I want to view a full memorandum (correction and explanation) for each practice question at the end so that I can learn from my mistakes.

10. As a student, I want to retake problems I previously got wrong so that I can reinforce learning and improve on my weaknesses.

11. As a student, I want to earn badges and rewards as I complete goals and milestones so that I stay motivated.

12. As a student, I want to earn XP for correctly solving questions so that I feel rewarded for my effort.

13. As a student, I want to appear on a leaderboard ranked by XP, so that I can compare myself to others and engage in healthy competition.

14. As a student, I want to access the platform on both my phone and desktop so that I can practice anytime, anywhere.

2. System

Adaptive Testing & ELO Calculation

2. As the system, I want to assign difficulty levels to questions from 1 to 10 so that I can measure user performance accurately.

4. As the system, I want to update a student's ELO rating after each problem submission (especially in ranked mode) so that future challenges reflect their current skill level.

5. As the system, I want to select problems based on a student's ELO and topic history so that they receive appropriate and varied challenges.

6. As the system, I want to adapt question difficulty dynamically based on recent performance trends so that the challenge remains balanced.

7. As the system, I want to store and track each student's ELO history over time so that trends and improvement can be visualized.

Submission Evaluation & Feedback

8. As the system, I want to parse and evaluate math expressions submitted in LaTeX format so that I can accurately assess correctness.

9. As the system, I want to provide immediate visual feedback (correct/incorrect, color indicators) after a submission so that students can learn effectively.

10. As the system, I want to suggest short tutorials or hints when a student fails a problem multiple times so that they don't feel stuck.

11. As the system, I want to recommend a review topic when a student repeatedly struggles in a specific area so that they can focus their improvement.

Tracking & Analytics

12. As the system, I want to log every problem attempt with metadata (time taken, number of attempts, type of answer) so that analytics remain comprehensive.

13. As the system, I want to track the time of day and session duration so that I can optionally suggest breaks when students seem fatigued (wellness feature).

Leaderboard and Abuse Prevention

14. As the system, I want to update the leaderboard in real-time when a user's XP/ELO changes so that rankings are always current.

15. As the system, I want to detect abnormal behavior (e.g, spamming submissions, solving too fast) so that abuse or misuse of the platform can be flagged.

Notifications & Engagement

16. As the system, I want to send push notifications reminding students to complete their daily lesson so that they stay consistent in their learning routine.

17. As the system, I want to notify students when new badges or achievements are unlocked so that they feel recognised and motivated.

18. As the system, I want to remind students if they haven't logged in for several days so that they do not fall behind or forget to practice.

19. As the system, I want to notify students when their leaderboard position changes so that they stay engaged and motivated to keep progressing.

3. Admin (Developers and Stakeholders)

Content & Question Management

1. As an admin, I want to upload, manage, and categorize math problems by topic and difficulty so that the problem pool stays educational, relevant, and diverse.

2. As an admin, I want to update or delete outdated questions so that users always receive accurate content.

3. System Requirements

FR1: User Registration and Profile Creation

- **FR1.1** The system must provide a registration form for students to input name, surname, age, email address, grade and math confidence level.
    - FR1.1.1: If a username that is already selected is chosen, prevent the student from selecting that username.
- **FR1.2** The system shall validate and securely store user information.
- **FR1.3** The system shall allow students to edit their profile data after registration.

FR2: Secure Login and Authentication

- **FR2.1** The system shall allow students to log in using their username and password.
- **FR2.2** The system shall implement password hashing and secure authentication mechanisms.
- **FR2.3** The system shall provide error messages for incorrect login attempts and allow password resets.

- **FR3.1** The system shall show one math question at a time to reduce cognitive load.
- **FR3.2** The system shall determine whether to branch left (easier) or right (harder) in the decision tree based on correctness of the student's answer.
- **FR3.3** The system shall use question difficulty levels ranging from 1 to 10.
- **FR3.4** The system shall compute an initial ELO rating based on consistent performance at a given level.

FR4: Game Modes and Math Practice

- **FR4.1** The system shall allow students to choose from three modes: Ranked Matches, Practice Rounds, and Single player Rounds.
- **FR4.2** The system shall deliver Ranked Match questions based on the user's current ELO.
- **FR4.3** The system shall limit Single Rounds to a predefined time and track the number of correct answers within that period.
- **FR4.4** The system shall update the student's ELO rating after every ranked match.
- **FR4.5** The system shall allow students to retake previously incorrect problems from their practice history.

FR5: Math Keyboard Input

- **FR5.1** The system shall include a math keyboard supporting symbols like fractions, exponents, square roots, and Greek letters.

FR6: Feedback and Memorandum

- **FR6.1** The system shall display immediate visual feedback (correct/incorrect) after every answer.
- **FR6.2** The system shall display a full memorandum or worked solution after each practice or test session.
- **FR6.3** The system shall recommend review topics or hints when a student struggles repeatedly.

FR7: XP, Badges, and Leaderboards

- **FR7.1** The system shall award XP for correct answers in Practice and Single player Rounds.
- **FR7.2** The system shall assign badges for milestone achievements.
- **FR7.3** The system shall maintain a real-time leaderboard ranked by XP.
  FR7.4 The system shall allow filtering of the leaderboard by ELO ranking.

FR8: ELO & Performance Analytics

- **FR8.1** The system shall display the student's current ELO rating and progress chart over time.

- **FR8.2** The system shall store a history of ELO changes and match data.
- **FR8.3** The system shall allow students to view accuracy, number of questions attempted, and topic mastery.

FR9: Push Notifications

- **FR9.1** The system shall notify students to complete their daily lesson or activity.
- **FR9.2** The system shall notify students when badges or achievements are unlocked.
- **FR9.3** The system shall notify inactive users after a predefined period.
- **FR9.4** The system shall notify users when their leaderboard rank changes.
- **FR9.5** The system shall notify students when their account is deleted or deactivated by admin.

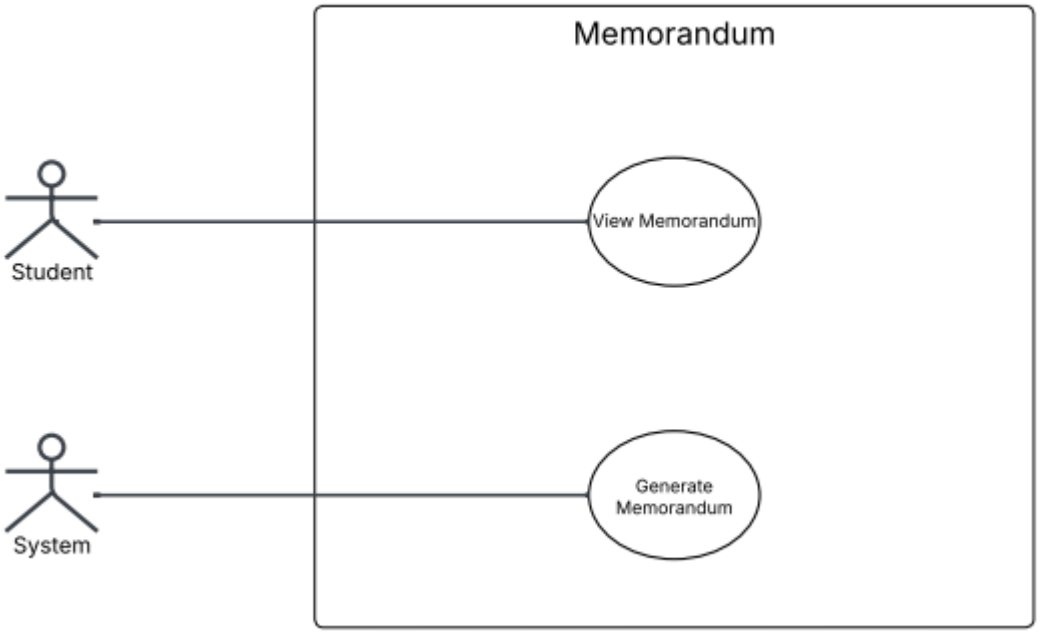
FR10: User Accessibility

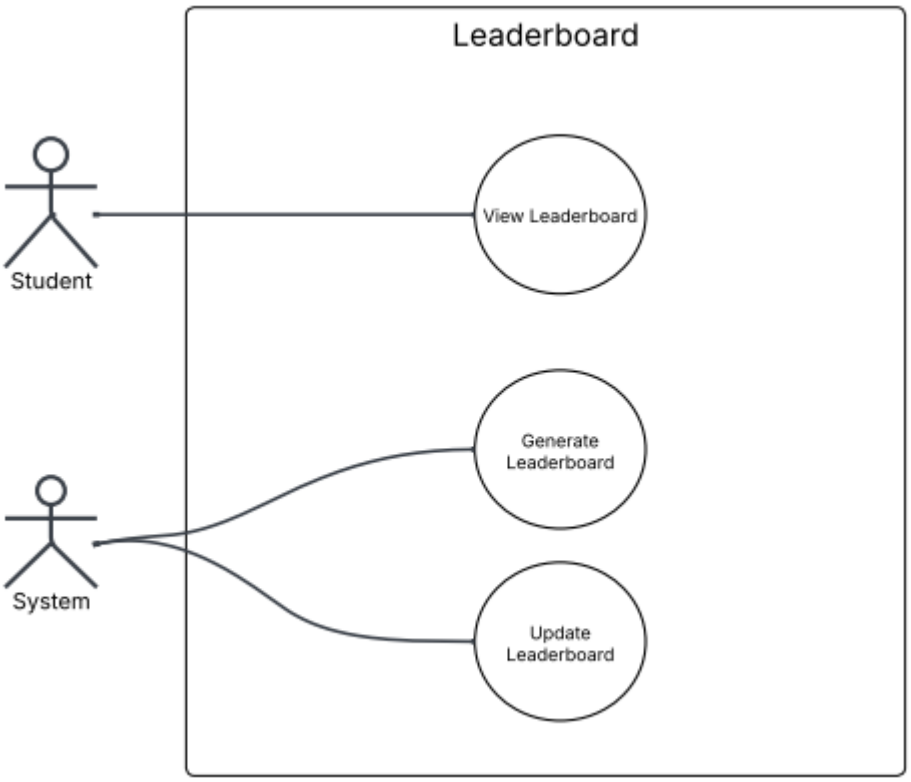
- **FR10.1** The system shall be accessible on desktop and mobile browsers.
- **FR10.2** The interface shall be responsive and support input from touchscreens and keyboards.

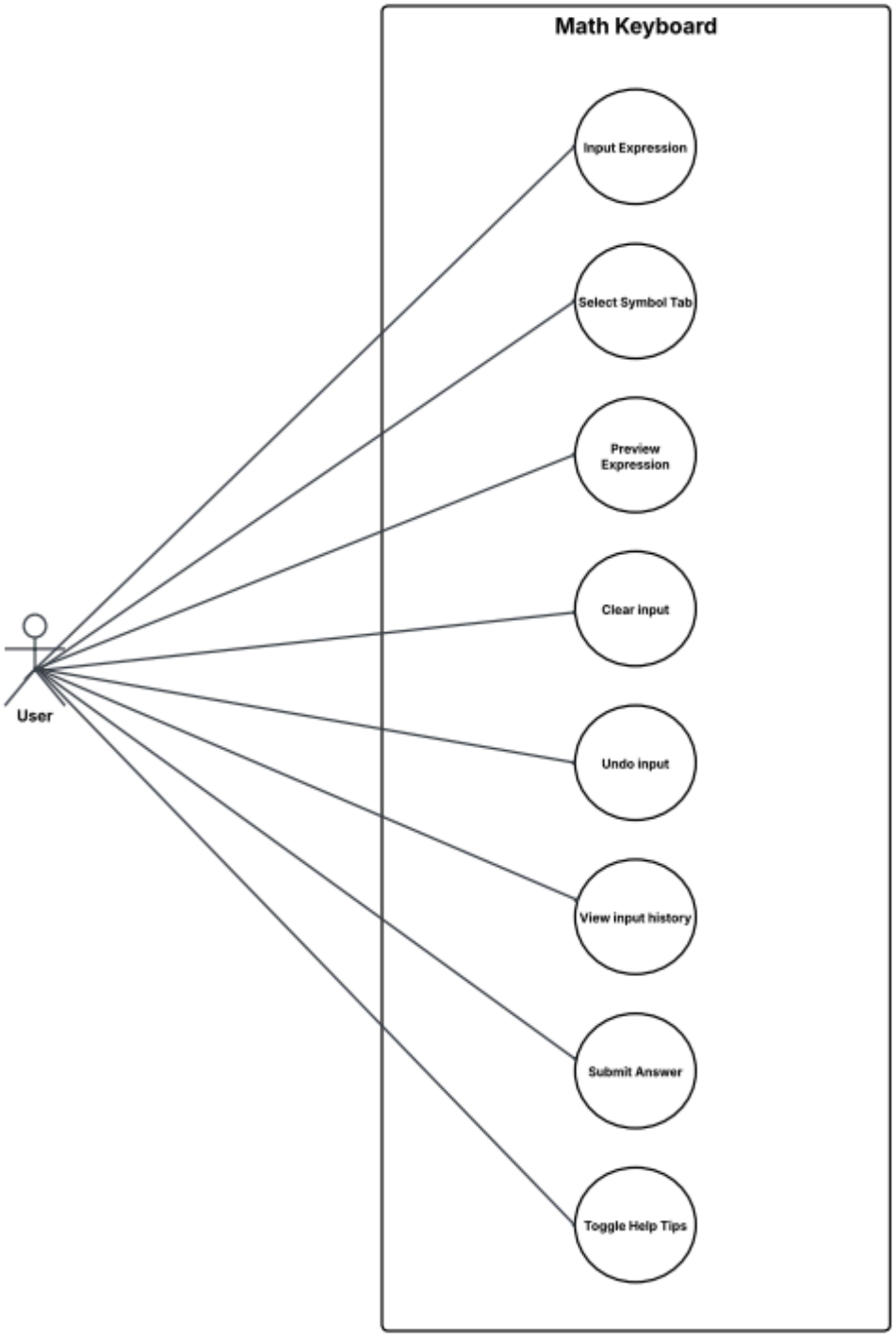
FR11: Admin – Content & Question Management

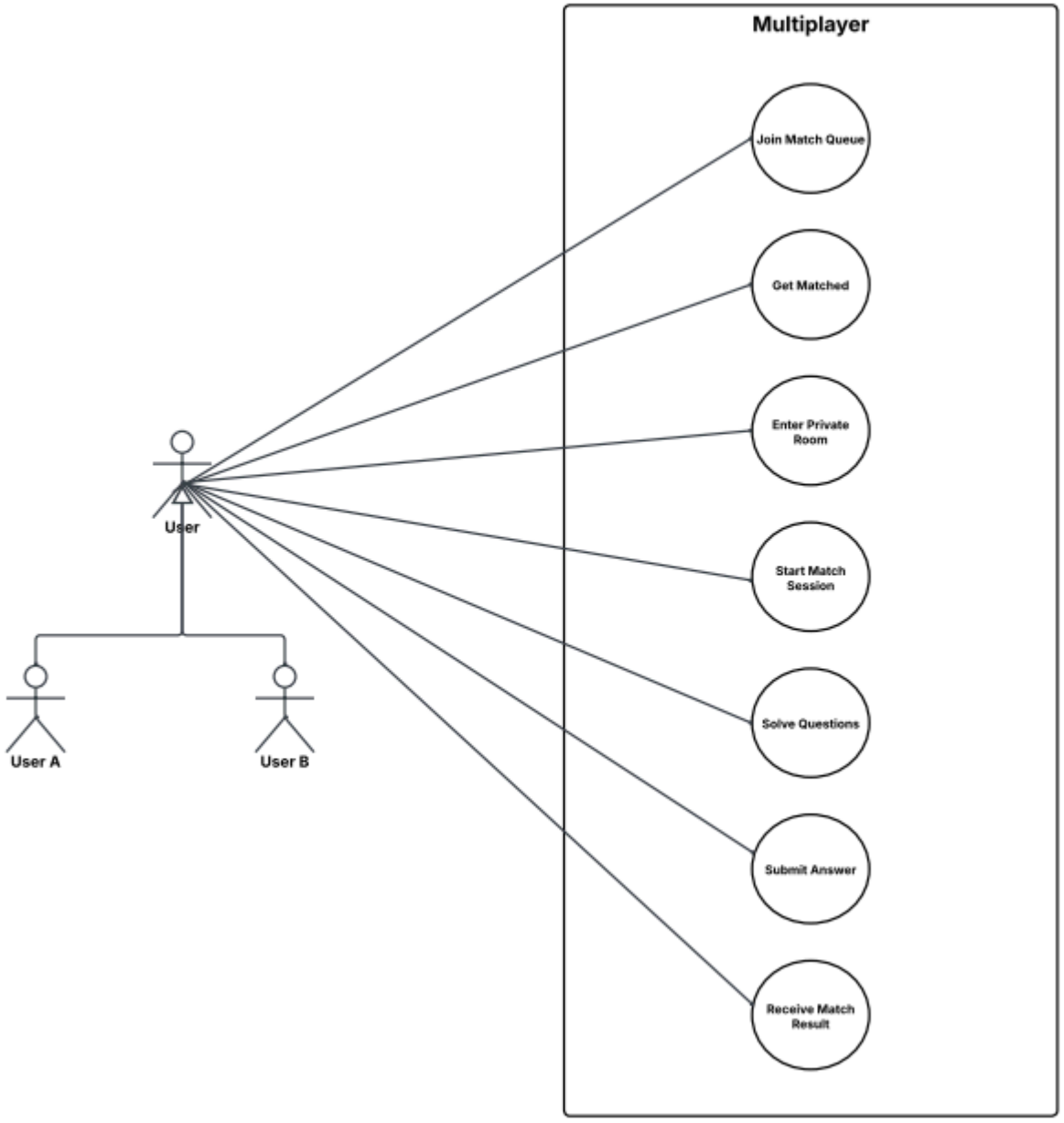
- **FR11.1** The system shall allow admins to upload, edit, delete, and categorize math problems by topic and difficulty.

4. Use Cases

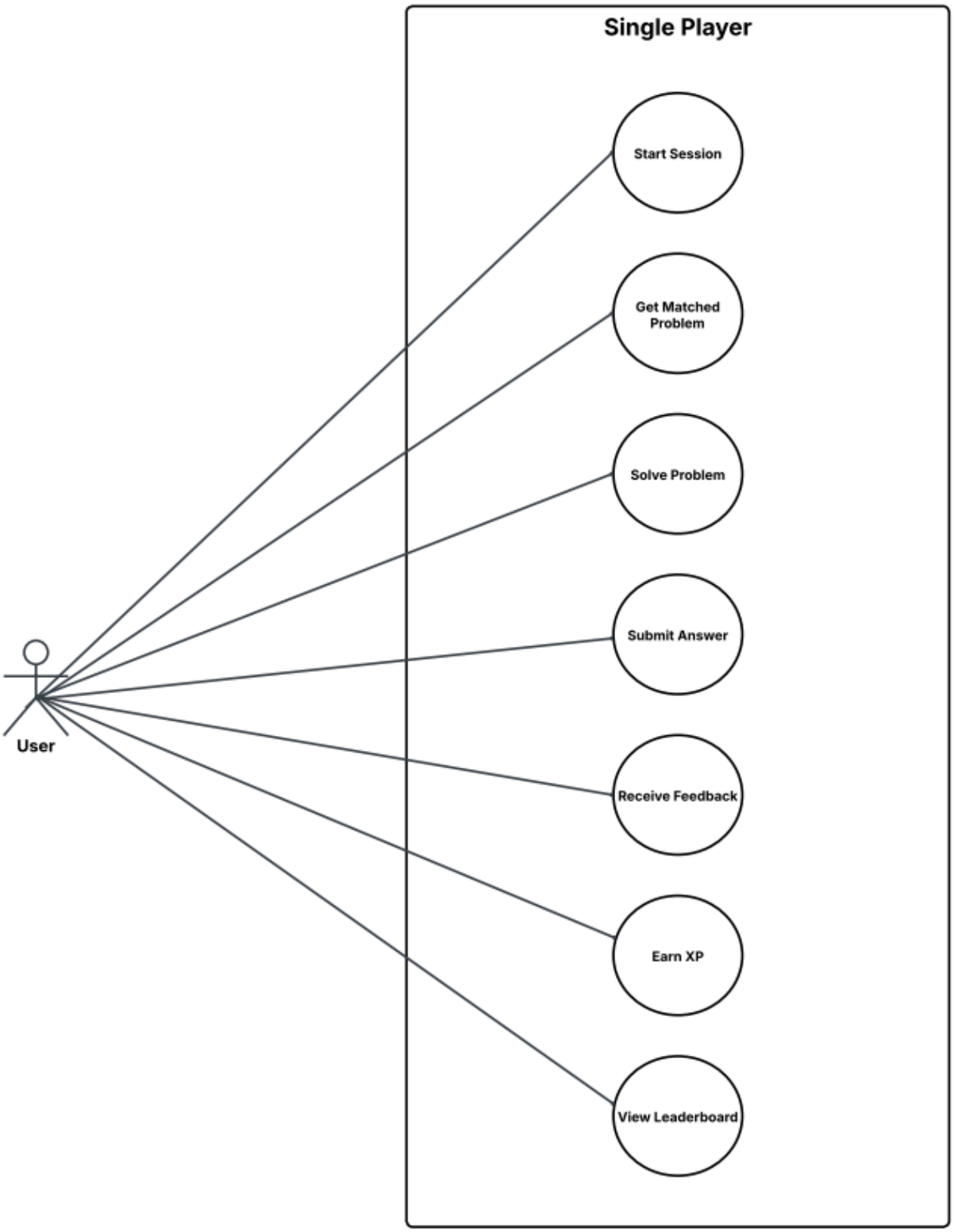

Memorandum
Student
View Memorandum
System
Generate Memorandum


Leaderboard
Student
View Leaderboard
Generate Leaderboard
System
Update Leaderboard

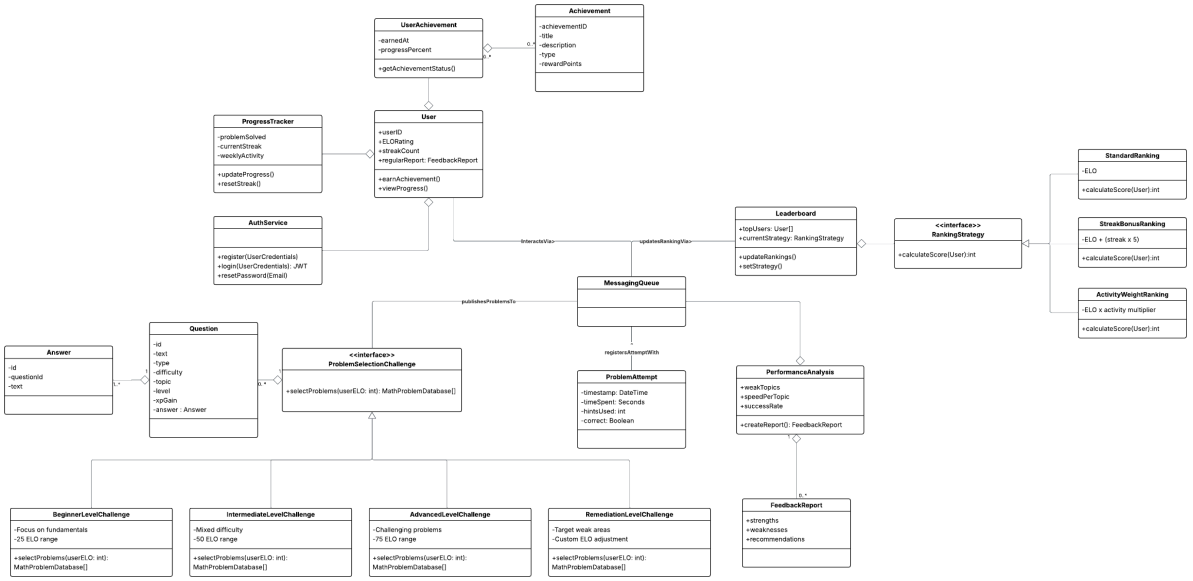

Math Keyboard
Input Expression
Select Symbol Tab
Preview Expression
Clear input
Undo input
View input history
Submit Answer
Toggle Help Tips
User


Multiplayer
Join Match Queue
Get Matched
Enter Private Room
Start Match Session
Solve Questions
Submit Answer
Receive Match Result
User
User A
User B


Practice
Choose Section
Answer Question
Submit Answer
Choose Table
Check Answer
Track Time
Assign XP
Generate Memorandum
Student
System


Single Player
Start Session
Get Matched Problem
Solve Problem
Submit Answer
Receive Feedback
Earn XP
View Leaderboard
User

5. Domain Model



6. Math Sections

Grade 8:

Paper 1	Paper 2
Fractions	Construction of Geometric Figures
Integers	Geometry of 2D Shapes
Exponents	Geometry of Straight Lines
Numeric and Geometric Patterns	
Functions and Relationships	
Algebraic Expressions	
Algebraic Equations	

Grade 9:

Paper 1	Paper 2
Fractions	Construction of Geometric Figures
Integers	Geometry of 2D Shapes
Exponents	Geometry of Straight Lines
Numeric and Geometric Patterns	Pythagoras' Theorem
Functions and Relationships	Area and Perimeter of 2D Shapes
Algebraic Expressions	
Equations	

Grade 10:

Paper 1	Paper 2
Fractions	Construction of Geometric Figures
Integers	Geometry of 2D Shapes
Exponents	Geometry of Straight Lines
Numeric and Geometric Patterns	Pythagoras' Theorem

Functions and Relationships	Area and Perimeter of 2D Shapes
Algebraic Expressions	
Equations	

Grade 11 + 12:

Paper 1	Paper 2
Algebra & equations	Statistics
Patterns & Sequences	Analytical geometry
Financial Mathematics	Trigonometry
Functions and Graphs	Euclidean geometry (Circle Geometry)
Probability	Measurement
Introduction to Calculus	

Technology Choices

Frontend Development: React.js and Next.js (PWA)

Alternatives Considered:

1. **Vue.js & Nuxt.js**: Strong SSR capabilities and gentle learning curve
2. **SvelteKit**: Excellent performance with minimal bundle size
3. **React.js & Next.js**: (Selected) Mature ecosystem with comprehensive PWA support

Selection Justification: React + Next.js provides the best balance of development velocity, PWA capabilities, and ecosystem support for complex mathematical interfaces. The mature component ecosystem includes specialized math rendering libraries that directly support our usability quality requirements.

Backend Development: Express.js

Alternatives Considered:

1. **Express.js**: (Selected) Lightweight, flexible Node.js framework with extensive middleware ecosystem
2. **NestJS**: Structured TypeScript framework with built-in dependency injection and testing
3. **Spring Boot (Java)**: Enterprise-grade framework with comprehensive features but steeper learning curve

Selection Justification: Express.js was chosen for its simplicity and rapid development capabilities, which align perfectly with our project timeline and team expertise. While NestJS offers more structure through its opinionated architecture, Express.js provides the flexibility needed to implement our SOA pattern without the overhead of learning a complex framework. The extensive middleware ecosystem allows us to add exactly the features we need for authentication, WebSocket support, and API routing without unnecessary complexity. This choice directly supports our performance quality requirements through minimal overhead and our maintainability requirements through the team's existing familiarity with Express.js patterns. The framework's lightweight nature also supports our scalability goals by reducing resource consumption per service instance.

Database Strategy: PostgreSQL + InfluxDB

Alternatives Considered:

1. **MongoDB + PostgreSQL**: NoSQL flexibility with relational consistency
2. **MySQL + Prometheus**: Standard relational with monitoring-focused time-series
3. **PostgreSQL + InfluxDB**: (Selected) Robust relational with specialized time-series capabilities

Selection Justification: PostgreSQL provides the ACID compliance needed for user data and ELO calculations, while InfluxDB offers optimized time-series storage for learning analytics. This combination directly supports our performance and scalability quality requirements.

Real-Time Communication: NestJS WebSocket Gateway

Alternatives Considered:

1. **Socket.IO (standalone)**: Feature-rich but requires additional integration overhead
2. **Firebase Realtime Database**: Easy setup but vendor lock-in concerns
3. **NestJS WebSocket Gateway**: (Selected) Integrated with existing backend architecture

Selection Justification: Native integration with our SOA services eliminates additional complexity while providing the <300ms latency required by our performance quality requirements.

Authentication: OAuth 2.0 + JWT

Alternatives Considered:

1. **Firebase Auth**: Simplified implementation but vendor dependency
2. **Session-based Auth**: Traditional approach but limited scalability
3. **OAuth 2.0 + JWT**: (Selected) Industry standard with scalable token-based architecture

Selection Justification: Provides the security requirements while supporting our SOA pattern's stateless service communication. The standard approach ensures long-term maintainability and compliance with security best practices.

[Services Contracts]

Establishing contracts between frontend and backend

Key terms

Frontend	Also known as the React PWA application.
Backend	This includes API and Database, and any layers in between
DB	Database

Users **3**
User object: 3
GET /users 3
URL Params 3
Data Params 3
Headers 3
Success Response 3
GET /user:id 4
URL Params 4
Data Params 4
Headers 4
Success Response 4
Error Response: 4
GET /users/:id/achievements 5
URL Params 5
Data Params 5
Headers 5
Success Response 5
Error Response: 5
POST /user/:id/xp 6
URL Params 6
Data Params 6
Headers 6
Success Response 6
Error Response: 6
Questions **7**
Question object: 7
GET /questions 7
URL Params 7
Data Params 7
Headers 7
Success Response 7
GET /question/:level 8
URL Params 8
Data Params 8
Headers 8
Success Response 8
Error Response: 8
GET /question/:id/answer 9
URL Params 9
Data Params 9
Headers 9

Success Response 9
Error Response: 9
GET /questions/topic 10
URL Params 10
Data Params 10
Headers 10
Success Response 10
Contents 10
GET /questions/level/topic 11
URL Params 11
Headers 11
Data Params 11
Success Response 11
POST /question/:id/answer 12
URL Params 12
Data Params 12
Headers 12
Success Response 12
Contents
Practice Endpoints
GET/practice
URL Params 12
Data Params 12
Headers 12
Success Response 12
Contents
Answer Endpoints
POST /question/:id/answer 12
URL Params 12
Data Params 12
Headers 12
Success Response 12
Contents
XP and Multiplayer Endpoints
POST /question/:id/answer 12
URL Params 12
Data Params 12
Headers 12
Success Response 12
Contents 12

Users

User object:

```
{
      id: integer
      name: string
      surname: string
      username: string
      email: string
      password: string
      currentLevel: integer
      joinDate: date
      xp: float
}
```

GET /users

Returns all users in the database.

URL Params

None

Data Params

None

Headers

Content-Type: application/json

Success Response

Code: 200

Contents

```
{
  users: [
           {<user_object>},
           {<user_object>},
           {<user_object>}
         ]
}
```

GET /user:id

Returns the specified user.

URL Params

Required: id=[integer]

Data Params

None

Headers

Content-Type: application/json

Authorization: Bearer <OAuth Token>

Success Response

Code: 200

Contents

```
{ <user_object> }
```

Error Response:

Code: 404

Content:

```
{ error : "User doesn't exist" }
```

OR

Code: 401

Content:

```
{ error : error : "You are unauthorized to make this request." }
```

GET /users/:id/achievements

Returns all achievements specific to a user.

URL Params

Required: id=[integer]

Data Params

None

Headers

Content-Type: application/json

Authorization: Bearer <OAuth Token>

Success Response

Code: 200

Content:

```
{
  achievements: [
            {<achievement_object>},
            {<achievement_object>},
            {<achievement_object>}
          ]
}
```

Error Response:

Code: 404

Content:

```
{ error : "User doesn't exist" }
```

OR

Code: 401

Content:

```
{ error : error : "You are unauthorized to make this request." }
```

POST /user/:id/xp

Updates the user's xp.

URL Params

Required: id=[integer]

Data Params

{
    id: integer,
    xp: float
}

Headers

Content-Type: application/json
Authorization: Bearer <OAuth Token>

Success Response

Code: 200
Content:
`{ <user_object> }`

Error Response:

Code: 404
Content:
`{ error : "User doesn't exist" }`

OR

Code: 401
Content:
`{ error : error : "You are unauthorized to make this request." }`

Questions

Question object:

```
{
      Q_id: integer
      topic: string
      difficulty: string
      level: integer
      questionText: string
      xpGain: float
}
```

GET /questions

Gets all the questions from the database.

URL Params

None

Data Params

None

Headers

Content-Type: application/json

Success Response

Code: 200

Contents

```
{
  questions: [
            {<question_object>},
            {<question_object>},
            {<question_object>}
          ]
}
```

GET /question/:level

Gets all questions based on the input level.

URL Params

Required: level=[integer]

Data Params

None

Headers

Content-Type: application/json

Authorization: Bearer <OAuth Token>

Success Response

Code: 200

Contents

```
{
  questions: [
            {<question_object>},
            {<question_object>},
            {<question_object>}
          ]
}
```

Error Response:

Code: 404

Content:

```
{ error : "Level doesn’t exist" }
```

OR

Code: 401

Content:

```
{ error : error : "You are unauthorized to make this request." }
```

GET /question/:id/answer

Gets the answer to a specific question.

URL Params

Required: Q_id=[integer]

Data Params

None

Headers

Content-Type: application/json

Authorization: Bearer <OAuth Token>

Success Response

Code: 200

Contents

```
{
  answer: [
            {<answer_object>},
          ]
}
```

Error Response:

Code: 404

Content:

```
{ error : "Question doesn't exist" }
```

OR

Code: 401

Content:

```
{ error : error : "You are unauthorized to make this request." }
```

GET /questions/topic

Returns all of the questions for that specific topic

URL Params

Required: topic="topic_tag"

Data Params

None

Headers

Content-Type: application/json

Success Response

Code: 200

Contents

```
{
  questions: [
            {<question_object>},
            {<question_object>},
            {<question_object>}
          ]
}
```

GET /questions/level/topic

Get all of the questions based on their level and their topic.

URL Params

Required: level=[integer]

Required: topic="topic_tag"

Headers

Content-Type: application/json

Data Params

```
{
  answer: string
}
```

Success Response

Code: 200

Contents

```
{
  result: boolean
}
```

POST /question/:id/answer

Send the question's answer through to be checked if correct.

URL Params

Required: Q_id=[integer]

Data Params

```
{
  question: [
            {<question_object>},
            {<question_object>},
            {<question_object>}
          ]
}
```

Headers

Content-Type: application/json

Success Response

Code: 200

Contents

```
{
  questions: [
            {<question_object>},
            {<question_object>},
            {<question_object>}
          ]
}
```

GET /questionsById/:id

Fetches a specific question by its ID.

URL Params

Required: `id=[integer]`

Headers

Content-Type: application/json

Success Response

Code: 200

Contents:

```
{
  "question": { <question_object> }
}
```

GET /answers/:id

Returns all answers for a specific question.

URL Params
Required: `id=[integer]`

Headers
Content-Type: application/json

Success Response
Code: 200
Contents:

```
{
  "answers": [
    { <answer_object> },
    { <answer_object> }
  ]
}
```

Practice Endpoints

GET /practice

Returns 10 practice questions and their answers.

Headers
Content-Type: application/json

Success Response
Code: 200
Contents:

```
{
  "questions": [
    { <question_object_with_answers> },
    { <question_object_with_answers> }
  ]
}
```

GET /practice/type/:questionType

Returns 10 questions of a specific type and their answers.

URL Params
Required: `questionType=[string]` (e.g., `"multiple-choice"` or `"true-false"`)

Headers
Content-Type: application/json

Success Response
Code: 200
Contents:

```
{
  "questions": [
    { <question_object_with_answers> },
    { <question_object_with_answers> }
  ]
}
```

Answer Endpoints

POST /submit-answer

Checks if the selected answer is correct and awards XP.

Headers
Content-Type: application/json

Data Params

```
{

  "questionId": [integer],

  "selectedAnswer": "[string]"

}
```

Success Response
Code: 200
Contents:

```
{

  "correct": true,

  "xpAwarded": [integer]

}
```

POST /question/:id/submit

Validates user answers to specific questions and awards XP**.**

URL Params
Required: `id=[integer]`

Headers
Content-Type: application/json

Data Params

```
{

  "answers": [ "[string]", "[string]" ]
```

```
}
```

Success Response
Code: 200
Contents:

```
{
  "results": [
    { "correct": true, "xp": 10 },
    { "correct": false, "xp": 0 }
  ]
}
```

POST /validate-answer

Validates if math answers match.

Headers
Content-Type: application/json

Data Params

```
{
  "expected": "[string]",
  "submitted": "[string]"
}
```

Success Response
Code: 200
Contents:

```
{
  "isValid": true
}
```

POST /quick-validate

Performs quick validation of math answers.

Headers
Content-Type: application/json

Data Params

```
{

  "expected": "[string]",

  "input": "[string]"

}
```

Success Response
Code: 200
Contents:

```
{

  "valid": true

}
```

POST /validate-expression

Checks if a math expression is valid.

Headers
Content-Type: application/json

Data Params

```
{

  "expression": "[string]"

}
```

Success Response
Code: 200
Contents:

```
{

  "valid": true
```

```
}
```

XP and Multiplayer Endpoints

POST /singleplayer

Handles XP calculations for a single player answering a question.

Headers
Content-Type: application/json

Data Params

```
{

  "userId": [integer],

  "questionId": [integer],

  "correct": true

}
```

Success Response
Code: 200
Contents:

```
{

  "xpAwarded": 10,

  "newTotalXP": 150

}
```

POST /multiplayer

Handles XP and match processing for multiplayer mode.

Headers
Content-Type: application/json

Data Params

```
{
```

```
  "player1": {
    "userId": [integer],
    "score": [integer]
  },
  "player2": {
    "userId": [integer],
    "score": [integer]
  },
  "questionId": [integer]
}
```

Success Response
Code: 200
Contents:

```
{
  "matchResult": "player1_wins",
  "xpUpdates": {
    "player1": 20,
    "player2": 5
  }
}
```

Architecture Requirements

ELO Learning - Architectural Requirements v2

Quality Requirements

QR1: Usability

- QR1.1 The system shall have a responsive UI that adjusts seamlessly across desktop, tablet, and mobile devices with 100% viewport compatibility testing.
- QR1.2 The interface shall adhere to WCAG 2.1 AA accessibility standards, including keyboard navigation and color contrast compliance with a minimum contrast ratio of 4.5:1.
- QR1.3 The platform shall provide visual cues (color indicators, feedback messages) to enhance user understanding with a measurable task completion rate of >85%.
- QR1.4 Onboarding tutorials shall achieve >90% completion rate and reduce time-to-first-successful-problem-attempt to under 3 minutes.

SOA Support for Usability: Our service-oriented architecture directly enhances usability through service specialization and loose coupling. The User Profile Service maintains consistent user preferences and progress across all interactions, while the Math Problem Service ensures mathematical content renders uniformly regardless of which frontend component requests it. Service interfaces provide standardized response formats that enable consistent user experience patterns across different platform areas. The separation of concerns allows the frontend to focus entirely on user experience optimization without being constrained by backend complexity, while services can evolve their internal implementations to better support usability requirements without affecting other system components

QR2: Performance

- QR2.1 The system shall maintain a median page load time of under 2 seconds for all major views measured via synthetic monitoring.
- QR2.2 The system shall support real-time updates using WebSockets with latency <300ms under normal load (up to 1,000 concurrent users).
- QR2.3 Cached content shall reduce server requests by 70% for static resources and frequently accessed data.

SOA Support for Performance: The service-oriented architecture enables performance optimization through independent service scaling and specialized resource management. The Stats/Leaderboard Service can be scaled independently during peak usage periods without affecting the Auth Service or Math Problem Service performance. Service-level caching strategies allow each service to optimize its data access patterns—the Math

Problem Service implements aggressive caching for frequently accessed problems, while the Matchmaking Service caches ELO calculations to reduce computational overhead. The loose coupling between services prevents performance bottlenecks in one service from cascading to others, and service-specific database optimization ensures that each service can tune its data access patterns for optimal performance without constraints from other system components.

QR3: Scalability

- QR3.1 The backend infrastructure shall support horizontal scaling to handle at least 10,000 concurrent users without >10% performance degradation.
- QR3.2 Services shall be independently scalable based on load patterns with automatic scaling triggers at 70% resource utilization.
- QR3.3 Load balancing shall distribute requests with <5% variance across available service instances.

SOA Support for Scalability: Service-oriented architecture provides exceptional scalability advantages through independent service scaling and resource allocation. Each service can be scaled based on its specific usage patterns—during peak learning hours, the Matchmaking Service may require more instances to handle problem assignment requests, while the Analytics Service can maintain minimal resources until batch processing periods. The service boundaries prevent resource contention, allowing high-demand services to scale without affecting the resource allocation of stable services like the Auth Service.
Service-level load balancing ensures that scaling decisions for one service don't impact the availability or performance of other services, and the shared database approach allows services to scale their computational resources independently while maintaining data consistency across the platform.

QR4: Reliability

- QR4.1 The system shall maintain 99.5% uptime per month, excluding planned maintenance windows.
- QR4.2 The system shall recover from service failure within 60 seconds using health checks and circuit breaker patterns.
- QR4.3 Daily automated backups shall maintain 99.9% data integrity with point-in-time recovery capability.

SOA Support for Reliability: The service-oriented architecture enhances system reliability through fault isolation and service independence. When individual services experience issues, the failure is contained within service boundaries—if the Analytics Service encounters problems, students can continue learning through the Math Problem Service and Matchmaking Service without interruption. Each service implements its own health monitoring and recovery mechanisms, allowing targeted recovery procedures that don't require full system restarts. The service interfaces provide standardized error handling that enables graceful degradation—if the Leaderboard Service is temporarily unavailable, the core learning functionality continues while users receive appropriate feedback about temporarily unavailable features. Service-level backup strategies ensure that critical services like Auth and User Profile have priority recovery procedures, while less critical services can be restored without impacting core educational functionality.

QR5: Security

- QR5.1 All HTTP requests shall be transmitted over HTTPS using TLS 1.3+.
- QR5.2 Authentication shall follow OAuth 2.0 + JWT standard with 1-hour access tokens and 7-day refresh tokens.
- QR5.3 User passwords shall be hashed using bcrypt with minimum 12 salt rounds.
- QR5.4 All API endpoints shall enforce role-based access control with Bearer Token authorization.
- QR5.5 POPIA compliance through user data management capabilities and consent mechanisms.

SOA Support for Security: Service-oriented architecture provides robust security advantages through service-level access control and security boundary enforcement. The Auth Service acts as a centralized security authority, issuing JWT tokens that other services validate independently, creating a consistent security model across all platform interactions. Each service implements role-based access control tailored to its specific domain—the Math Problem Service restricts problem editing to administrators while allowing read access to authenticated students, and the User Profile Service ensures users can only access their own progress data. Inter-service communication follows secure protocols with service-to-service authentication, preventing unauthorized access between internal components. The service boundaries create security compartmentalization where a potential vulnerability in one service doesn't automatically compromise other services, and service-specific security auditing enables targeted security monitoring and incident response procedures.

QR6: Maintainability

- QR6.1 All backend services shall follow consistent NestJS Controller-Service-Repository pattern.
- QR6.2 Code coverage shall maintain a minimum 80% measured via Jest testing framework.
- QR6.3 JSDoc documentation shall be provided for all public functions and modules.

SOA Support for Maintainability: Service-oriented architecture provides exceptional maintainability advantages through clear separation of concerns and modular development practices. Each service maintains its own codebase with well-defined boundaries, allowing developers to understand, modify, and extend individual services without needing to comprehend the entire system complexity. When modifications are required for the ELO matching algorithm, developers can focus exclusively on the Matchmaking Service without worrying about unintended effects on the Auth Service or User Profile Service. The service interfaces act as contracts that prevent breaking changes from propagating across service boundaries, enabling confident refactoring within individual services.

Service-level maintainability is further enhanced through independent deployment and versioning capabilities. Each service can evolve its internal implementation, adopt new libraries, or optimize its algorithms without coordinating changes across the entire platform. The shared database approach, while creating some coupling, actually improves

maintainability for educational platforms by ensuring data consistency and simplifying backup and recovery procedures. Service-specific documentation and testing strategies mean that maintenance work can be performed by developers who specialize in particular domains—authentication experts can maintain the Auth Service while education specialists focus on the Math Problem Service, leading to higher quality maintenance and more effective bug resolution.

QR7: Testability

- QR7.1 Backend services shall include unit and integration tests with >80% code coverage.
- QR7.2 E2E tests shall cover critical user journeys using Cypress with >95% test pass rate.
- QR7.3 CI pipelines shall fail builds when coverage drops below thresholds or critical tests fail.

SOA Support for Testability: Service-oriented architecture significantly enhances testability through service isolation and interface standardization. Each service can be unit tested independently using mock implementations of dependent services, enabling parallel test development and faster test execution. Service interface testing validates API contracts between services, ensuring that changes to one service don't break dependent services without explicit interface versioning. The architecture supports comprehensive integration testing strategies including service-to-service contract testing, where each service interface is tested against defined contracts to ensure compatibility. Mock service implementations allow isolated testing of individual services without requiring the full system to be operational, enabling efficient development and debugging cycles.

Service Testing Strategies: Our SOA implementation employs multiple testing layers tailored to service-oriented concerns. **Service Interface Testing** validates that each service correctly implements its API contracts and handles edge cases appropriately. **Mock Service Strategy** provides lightweight service doubles for unit testing, allowing developers to test service logic without external dependencies. **API Contract Testing** between services ensures backward compatibility and validates that service interfaces meet their specifications. **Service Integration Testing** verifies that services work correctly together, including authentication flows, data consistency, and error handling across service boundaries. **End-to-End Service Chain Testing** validates complete user workflows that span multiple services, ensuring that the service orchestration delivers expected business outcomes.

Architectural Design Strategy

For ELO Learning, we have chosen a **design strategy based on quality requirements** as our primary architectural approach. This strategy prioritizes the systematic achievement of our most critical quality attributes—usability, performance, and scalability—which directly impact the educational effectiveness of our platform.

Justification for Quality-Driven Design Strategy

The quality-driven approach is most suitable for ELO Learning because educational platforms must excel in user experience and performance to maintain student engagement. Unlike purely functional-driven design, this strategy ensures that architectural decisions directly support measurable learning outcomes. The strategy involves:

1. **Quality Attribute Scenarios**: Each quality requirement is expressed as testable scenarios that drive architectural decisions
2. **Tactic Selection**: Specific architectural tactics are chosen to address quality attribute requirements systematically
3. **Pattern Application**: Architectural patterns are selected based on their ability to satisfy our prioritized quality attributes
4. **Iterative Refinement**: The architecture evolves through continuous measurement against quality targets

Architectural Strategies

Our architectural strategies directly address our quality requirements through specific tactics and techniques:

Performance Strategy: Asynchronous Processing and Caching

- **Tactics**: Manage resources, increase available resources, reduce overhead
- **Implementation**: WebSocket-based real-time updates, Redis caching layer, CDN for static assets
- **Quality Target**: <2 second page load times, <300ms real-time update latency

Scalability Strategy: Horizontal Scale-out with Load Distribution

- **Tactics**: Multiple copies of data, multiple copies of computation
- **Implementation**: Container-based service deployment, horizontal pod autoscaling, database read replicas
- **Quality Target**: Support 10,000+ concurrent users with linear scaling

Usability Strategy: Real-time UI Responsiveness

- **Tactics**: Maintain task model, maintain system model, maintain user model
- **Implementation**: Progressive Web App architecture, responsive design patterns, accessibility-first development
- **Quality Target**: >85% task completion rate, <3 minute onboarding time

Security Strategy: Defense in Depth with Token-based Authentication

- **Tactics**: Authenticate users, authorize users, maintain data confidentiality
- **Implementation**: OAuth 2.0 + JWT, HTTPS/TLS 1.3, RBAC, input validation
- **Quality Target**: Zero security incidents, full POPIA compliance

Availability Strategy: Fault Detection and Recovery

- **Tactics**: Fault detection, fault recovery, fault prevention
- **Implementation**: Health checks, circuit breakers, automated failover, backup systems
- **Quality Target**: 99.5% uptime with <60 second recovery time

Architectural Patterns

Primary Pattern: Service-Oriented Architecture (SOA)

ELO Learning employs a **Service-Oriented Architecture (SOA)** pattern as its primary architectural approach. This represents an evolution from our initial microservices consideration, adapted to better suit our project's specific constraints and requirements.

Migration from Microservices to SOA: Justification

Initially, we considered a pure microservices architecture for its benefits of independent deployment and technology diversity. However, after careful analysis of our project constraints and team capabilities, we transitioned to SOA for the following reasons:

Complexity Management: Microservices proved too complex for our team size and timeline. The overhead of managing independent deployments, service discovery, distributed monitoring, and inter-service communication protocols would have consumed significant development time better spent on core educational features.

Resource Constraints: True microservices require substantial DevOps infrastructure and monitoring capabilities. Our three-component Demo 2 requirement and limited infrastructure budget made SOA's shared deployment model more practical.

Integration Simplicity: SOA's shared database approach and simplified inter-service communication patterns reduce the "distributed system tax" that microservices impose, allowing us to focus on educational functionality rather than distributed systems engineering.

Team Expertise: Our team's existing experience with monolithic and service-based patterns made SOA a more natural progression than the leap to full microservices architecture.

SOA Implementation Details

Our SOA pattern provides the modularity benefits we need while maintaining manageable complexity:

Service Characteristics:

- Each service is modular and focused on a specific domain (Auth, Matchmaking, Problem Management, etc.)
- Services communicate through well-defined REST APIs and WebSocket connections
- Services share database access but maintain clear domain boundaries

- Services are deployed together but can be scaled independently through container orchestration

Service Inventory:

1. **Auth Service**: Handles user registration, login, JWT generation, and OAuth 2.0 flow
2. **Matchmaking Service**: Implements ELO-based algorithm for problem difficulty matching
3. **Math Problem Service**: Manages problem storage, retrieval, and metadata
4. **Stats/Leaderboard Service**: Computes rankings and delivers leaderboard data
5. **User Profile Service**: Manages personal data, progress tracking, and achievements
6. **Analytics Service**: Logs interaction metrics and performance data (planned)

Benefits Realized

- **Simplified Integration**: Shared database access reduces inter-service communication complexity
- **Manageable Deployment**: Services deploy together while maintaining logical separation
- **Development Velocity**: Team can work on different services without complex coordination
- **Quality Assurance**: Easier to implement comprehensive testing across service boundaries

Tradeoffs Accepted

- **Less Independence**: Services are not fully independent, requiring some coordination for changes
- **Shared Database**: Potential coupling through shared data schemas
- **Deployment Coupling**: Services deploy together, reducing deployment flexibility compared to microservices

Secondary Pattern: Model-View-Controller (MVC) for Frontend

The frontend implementation follows the **Model-View-Controller (MVC)** pattern, implemented through React's component architecture with Next.js:

Model: Application state management through React hooks and context, representing user data, problem state, and UI state **View**: React components that render the user interface, including the custom math keyboard and problem displays
Controller: Event handlers and business logic that coordinate between user interactions and state updates

This pattern supports our usability quality requirements by providing clear separation between presentation and logic, enabling consistent UI behavior and easier maintenance of the complex mathematical input interfaces.

Supporting Patterns

Observer Pattern: Implemented through WebSocket connections for real-time leaderboard updates and progress notifications. This pattern directly supports our performance quality requirements for <300ms real-time updates.

Strategy Pattern: Used in the matchmaking service to allow different ELO calculation strategies and problem selection algorithms. This supports future extensibility without architectural changes.

Service Layer Pattern: Consistently applied across all backend services using NestJS's Controller-Service-Repository structure, supporting our maintainability quality requirements.

Mediator Pattern: Applied through our API Gateway pattern, which coordinates communication between frontend and backend services while providing security and monitoring capabilities.

Architectural Constraints

Technical Constraints

- **Container Deployment**: All services must be containerized using Docker for consistent deployment across environments
- **Cloud Platform**: System must deploy to either AWS or Azure using infrastructure-as-code principles
- **Database Technology**: Must use PostgreSQL for relational data and InfluxDB for time-series analytics data
- **Security Standards**: Must implement HTTPS/TLS 1.3, OAuth 2.0 + JWT authentication, and POPIA compliance

Project Constraints

- **Demo Timeline**: Three components must be fully implemented by June 27, 2025, limiting architectural complexity
- **Team Size**: Architecture must be manageable by a small development team without dedicated DevOps engineers
- **Budget Limitations**: Infrastructure costs must remain within educational project constraints

Regulatory Constraints

- **POPIA Compliance**: User data handling must comply with South African privacy regulations
- **Educational Standards**: Math content and progress tracking must support pedagogical best practices
- **Accessibility Requirements**: Interface must meet WCAG 2.1 AA standards for inclusive education

Integration Constraints

- **Math Input Complexity**: Architecture must support complex mathematical notation input and rendering
- **Real-time Requirements**: Must support WebSocket connections for immediate feedback and collaborative features
- **Progressive Web App**: Must function as PWA for mobile accessibility without native app development

Technology Choices

Frontend Development: React.js and Next.js (PWA)

Alternatives Considered:

1. **Vue.js & Nuxt.js**: Strong SSR capabilities and gentle learning curve
2. **SvelteKit**: Excellent performance with minimal bundle size
3. **React.js & Next.js**: (Selected) Mature ecosystem with comprehensive PWA support

Selection Justification: React + Next.js provides the best balance of development velocity, PWA capabilities, and ecosystem support for complex mathematical interfaces. The mature component ecosystem includes specialized math rendering libraries that directly support our usability quality requirements.

Backend Development: Express.js

Alternatives Considered:

1. **Express.js**: (Selected) Lightweight, flexible Node.js framework with extensive middleware ecosystem
2. **NestJS**: Structured TypeScript framework with built-in dependency injection and testing
3. **Spring Boot (Java)**: Enterprise-grade framework with comprehensive features but steeper learning curve

Selection Justification: Express.js was chosen for its simplicity and rapid development capabilities, which align perfectly with our project timeline and team expertise. While NestJS offers more structure through its opinionated architecture, Express.js provides the flexibility needed to implement our SOA pattern without the overhead of learning a complex framework. The extensive middleware ecosystem allows us to add exactly the features we need for authentication, WebSocket support, and API routing without unnecessary complexity. This choice directly supports our performance quality requirements through minimal overhead and our maintainability requirements through the team's existing familiarity with Express.js patterns. The framework's lightweight nature also supports our scalability goals by reducing resource consumption per service instance.

Database Strategy: PostgreSQL + InfluxDB

Alternatives Considered:

1. **MongoDB + PostgreSQL**: NoSQL flexibility with relational consistency
2. **MySQL + Prometheus**: Standard relational with monitoring-focused time-series
3. **PostgreSQL + InfluxDB**: (Selected) Robust relational with specialized time-series capabilities

Selection Justification: PostgreSQL provides the ACID compliance needed for user data and ELO calculations, while InfluxDB offers optimized time-series storage for learning analytics. This combination directly supports our performance and scalability quality requirements.

Real-Time Communication: NestJS WebSocket Gateway

Alternatives Considered:

1. **Socket.IO (standalone)**: Feature-rich but requires additional integration overhead
2. **Firebase Realtime Database**: Easy setup but vendor lock-in concerns
3. **NestJS WebSocket Gateway**: (Selected) Integrated with existing backend architecture

Selection Justification: Native integration with our SOA services eliminates additional complexity while providing the <300ms latency required by our performance quality requirements.

Authentication: OAuth 2.0 + JWT

Alternatives Considered:

1. **Firebase Auth**: Simplified implementation but vendor dependency
2. **Session-based Auth**: Traditional approach but limited scalability
3. **OAuth 2.0 + JWT**: (Selected) Industry standard with scalable token-based architecture

Selection Justification: Provides the security requirements while supporting our SOA pattern's stateless service communication. The standard approach ensures long-term maintainability and compliance with security best practices.

Custom Math Keyboard Implementation

Architecture Decision

The custom math keyboard represents a critical architectural component that directly impacts our highest-priority quality requirement: usability. Rather than relying on external services or complex integrations, we've architected an integrated solution that provides seamless mathematical input within our educational platform.

Technology Stack

- **MathLive**: Provides the interactive math keyboard with LaTeX support, enabling complex mathematical notation input
- **KaTeX**: Handles real-time math rendering with superior performance compared to MathJax
- **math.js**: Enables backend expression evaluation and automated grading capabilities
- **React Integration**: Custom wrapper components that integrate mathematical input with our MVC frontend pattern

Architectural Benefits

This integrated approach supports multiple quality requirements simultaneously: it enhances usability through intuitive math input, improves performance through optimized rendering, and maintains security through controlled input validation. The architecture ensures that mathematical notation handling remains consistent across all problem types while supporting future extensibility for advanced mathematical concepts.

7. WOW Factors!

FR12: Classroom Wars (Gamified Competitions)

FR12.1 The system shall allow teachers to create a "Classroom War" match between groups of students or entire classes.
FR12.2 The system shall automatically match students within a Classroom War based on similar ELO ratings.
FR12.3 The system shall provide real-time progress tracking of each class or group during the competition.
FR12.4 The system shall award bonus XP, badges, and achievements to winners of Classroom Wars.
FR12.5 The system shall provide a leaderboard for Classroom Wars, visible to all participants.

FR13: Customer Support & Help Desk

FR13.1 The system shall provide an in-app customer support portal accessible from the user dashboard.
FR13.2 The system shall allow students, parents, or teachers to submit support tickets describing technical or account issues.
FR13.3 The system shall allow support staff to view, respond, and resolve tickets within an admin panel.
FR13.4 The system shall notify users via push notifications or email when their ticket status changes.
FR13.5 The system shall provide a searchable FAQ and knowledge base for common issues.

FR14: AI Assistance & Adaptive Learning

FR14.1 The system shall use AI to recommend personalized practice sets based on student weaknesses and learning history.
FR14.2 The system shall allow students to request AI-generated hints or step-by-step explanations for difficult problems.
FR14.3 The system shall allow AI to detect patterns of repeated mistakes and suggest targeted review topics.
FR14.4 The system shall provide teachers with AI-generated analytics on student progress and areas of struggle.
FR14.5 The system shall adapt question difficulty dynamically using AI predictions of user performance.

FR15: Location-Based Rankings

FR15.1 The system shall allow leaderboards to be filtered by country, region, school, or classroom.
FR15.2 The system shall determine a user's location using profile data or device geolocation.
FR15.3 The system shall allow students to compare their performance against peers in the same city, school, or region.
FR15.4 The system shall award location-based badges (e.g., "Top 10 in Johannesburg").
FR15.5 The system shall ensure location data privacy and allow students to opt-out of public rankings.

FR16: Enhanced Push Notifications (Engagement)

FR16.1 The system shall send personalized push notifications based on a student's activity patterns (e.g., "Finish your streak!").
FR16.2 The system shall notify users of new Classroom Wars or challenges issued by classmates.
FR16.3 The system shall allow students to customize notification preferences (frequency, type, and channel).
FR16.4 The system shall send AI-driven motivational messages (e.g., reminders of progress or encouraging quotes).
FR16.5 The system shall notify parents/guardians about their child's learning milestones if linked accounts exist.

FR17: Community & Social Learning

FR17.1 The system shall allow students to join or create communities (e.g., "Grade 10 Geometry Club").
FR17.2 The system shall allow community members to post questions, share solutions, and discuss math problems.
FR17.3 The system shall allow admins to moderate communities and remove inappropriate content.
FR17.4 The system shall provide community achievements and badges for active participation.
FR17.5 The system shall allow students to invite friends or classmates to join their community.

Summary

The revised architecture for ELO Learning represents a carefully balanced approach that prioritizes our critical quality requirements while remaining practical for our team and timeline constraints. The migration from microservices to SOA reflects architectural maturity—choosing the right tool for the job rather than following trends. Our quality-driven design strategy ensures that every architectural decision directly supports measurable educational outcomes, while our comprehensive technology stack provides the foundation for a scalable, maintainable learning platform.

The architecture successfully addresses the tension between educational effectiveness and technical complexity, providing a robust foundation that can evolve with our platform's growth while maintaining the performance and usability standards that educational success requires.

Appendix: Reference Documentation

Demo 1 Architectural Requirements

For historical context and to trace the evolution of our architectural thinking, the original architectural requirements document from Demo 1 can be referenced at: Demo 1 Architectural Requirements Specification

This original document provides insight into our initial architectural considerations and shows how our understanding has matured through the development process. The transition from our Demo 1 specifications to this current version demonstrates the iterative nature of architectural design and how real-world constraints and deeper understanding of quality requirements have shaped our final architectural decisions.

The comparison between these versions illustrates several key architectural learning points: how initial complexity assumptions were refined through practical experience, how quality requirements evolved from general statements to measurable specifications, and how our service-oriented approach emerged as the most suitable pattern for our specific project context and team capabilities.