Mathify

Technical Report M2

Küng Pascal

Mettler Micha

Zehnder Jonas

Gonçalves Rafael

Thayananthan Ragavan

ZHAW-School of Engineering

IT22tb_WIN

Software Project 3

Lectured by Nina Schnatz and Kurt Bleisch

Table of Contents

1 Intro

At the core of Mathify's[1] app development for schoolchildren lies a dedication to usability and user experience. From the beginning, the focus was on how to ensure and enhance effectiveness, efficiency, and user satisfaction. For the prototype, attention was given particularly – but not exclusively – to three of the seven essential requirements:

1. **Suitability for the task** (German: Aufgabenangemessenheit): The user interface must be straightforward, displaying only the necessary content. The app should require the fewest possible steps and user input for logging in, solving exercises and watching tutorials[2], considering children often have limited technical skills and should not face unnecessary complications.
2. **Conformity with user expectations** (German: Erwartungskonformität): It is critical that the app's terminology, behaviour, and task presentation remain consistent. This ensures that once a child is familiar with the design and interaction, they will instinctively know what to do next. This consistency is crucial as children may read slowly or have difficulty understanding complex terms and interactions.
3. **Error tolerance** (German: Fehlertoleranz): The app must communicate clearly what input is expected and check for incorrect entries.

As the development moves beyond the prototype phase, customizability (German: Individualisierbarkeit) will become a focus. Initially, the app uses school grade as a starting point for the type of exercises, enabling progression from basic to advanced levels. Later, extensive customization will be introduced such as challenge modes with time-limits or custom modes with individually tailored exercises made by teachers.

Furthermore, learning facilitation (German: Lernförderlichkeit) will also become a priority. The aim is to provide users, upon request, with information about the underlying mathematical concepts and easy access to further tutorials, including videos. This will enable them to grasp complex concepts while engaging with the app.

[1] See Glossary for a definition of this term.

[2] The teaching tutorial feature is not part of the prototype.

2 Use-Case-Model

In the definition of the use case model, these steps were followed:

2.1 Step 1: Define System Boundaries

Defining the system boundaries is a critical initial step in the development of Mathify. It is instrumental in establishing the project's scope and objectives. By identifying Mathify as the system boundary, focus is placed on creating relevant use cases and delineating the functions and actors that fall within the ambit of Mathify, as opposed to those that do not. Such clarity not only enhances the efficient use of resources but also ensures transparent communication with stakeholders.

2.2 Step 2: Identify all Actors

Primary Actor

A primary actor initiates a use case to achieve a specific goal and, consequently, receives the main benefit. In this scenario, the primary actors include:

1. Child[3]
2. Teacher

Supporting Actor

A supporting actor assists the system in managing a use case. In this scenario, the supporting actors include:

1. App Team[4]
2. Payment Service
3. Parents

Offstage Actor

An offstage actor is an additional stakeholder who does not directly interact with the system. In this scenario, the offstage actors include:

1. School
2. Department of Education
3. Tax Department

[3] See Glossary for a definition of this term.

[4] See Glossary for a definition of this term.

System Boundary Diagram

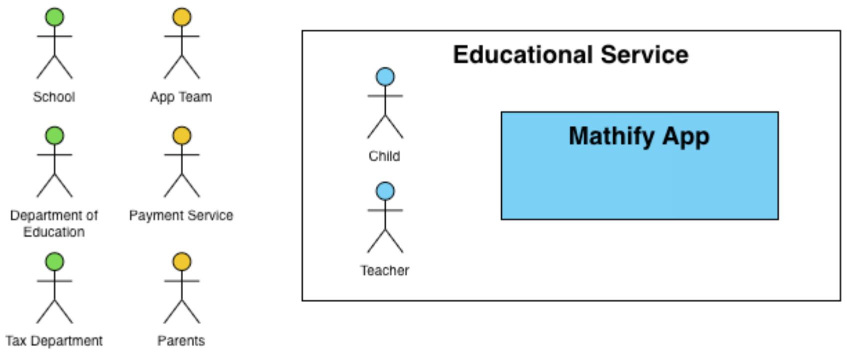


Figure 1: System Boundary Diagram

2.3 Step 3: Identify Goals and Tasks of Primary Actors

<u>Tasks</u>

- Solving math problems (child)
- Competing with others to improve (child)
- Creating exercises and mock exams (teacher)

<u>Goals</u>

- Enhancing math skills (child)
- Building self-esteem (child)
- Finding knowledge gaps and eliminating them (teacher)

2.4 Step 4: Define Use Cases

2.4.1 Register / Login of Child (Casual UC)

- Upon their initial login, a child, with or without parental assistance, is prompted to enter their username, password, grade, email address and payment details[5]. If the child already possesses an account, they are required to input their username and password. The system verifies all the information provided.
- The system offers a password reset option[6] for children who have forgotten their password, utilizing the email address for assistance.

[5] Payment details are not part of the prototype.

[6] The password reset option is not part of the prototype.

- Following successful login, the child selects a game mode[7], and the system identifies potential game statuses.
- The system then progresses to the exercise screen.

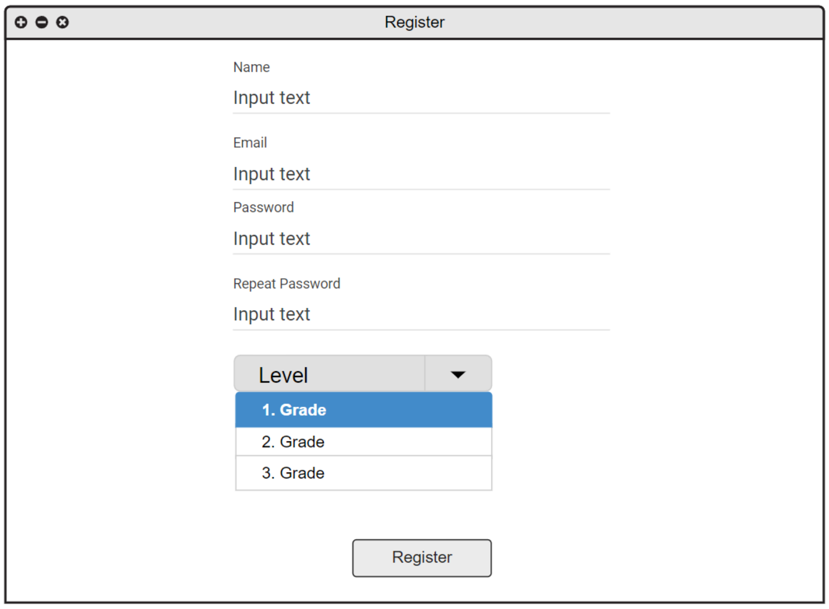


Figure 2: Mock-up Register-Screen

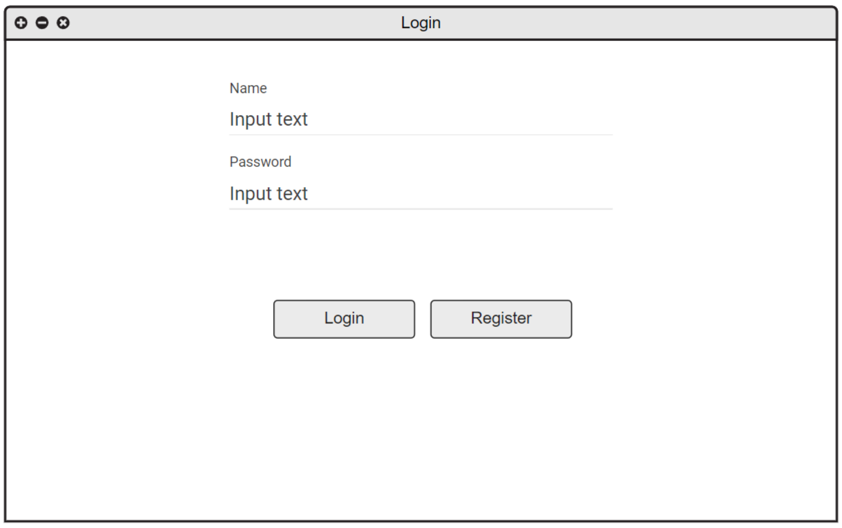


Figure 3: Mock-up Login-Screen

[7] See Glossary for a definition of this term.

2.4.2 Select Mode (Brief UC)[8]

- The child selects a mode. The system determines the type of exercise and further functionalities (e.g. time-constraints) based on the selection of the mode.

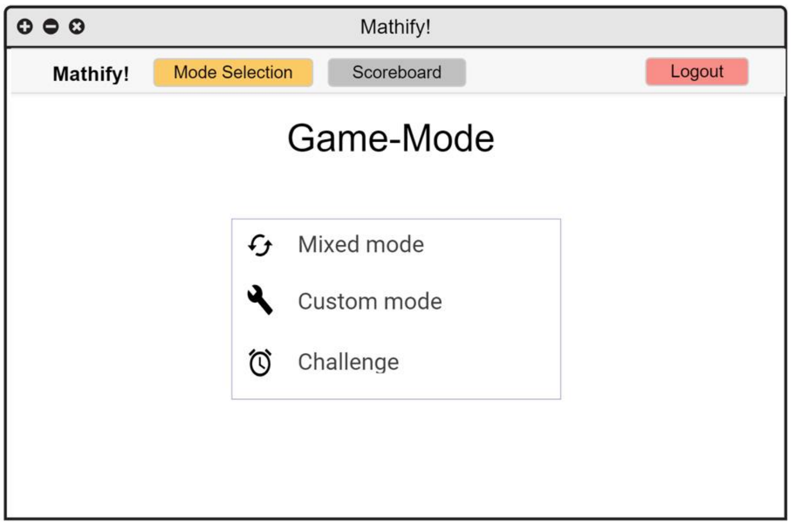


Figure 4: Mock-up Mode Selection

2.4.3 Solve Math Exercise (Fully Dressed UC)

Use Case Name:

- Solve Math Exercise

Scope:

- Mathematics Quiz

Level:

- User goal

Primary Actor:

- Child

[8] Mode selection is not part of the prototype.

Stakeholders and Interest:

- Teacher
    - Seeks to monitor the performance of their class.
    - Strives to identify and address any knowledge deficiencies.

- Parents
    - Aim to support their children's education.
    - Wish to keep track of their children's educational progress.

- Schools and Educational Department
    - Aim to raise educational standards across the nation.

- Tax Authority
    - Seeks to collect tax revenue.

Preconditions:

- The child must be properly logged in and licensed.

Success Guarantee / Postconditions:

- Exercises are generated accurately.
- Submissions are validated effectively.
- Feedback is given back to the child.
- Child 's current progress (level + experience) is stored for future gameplay.

Standard Process:

1. The system generates an exercise.
2. The child completes the exercise.
3. The system validates the child's submission.
4. The system verifies the correctness of the result.
5. The system provides feedback to the child.
6. The child repeats steps 1-5 to solve additional exercises.

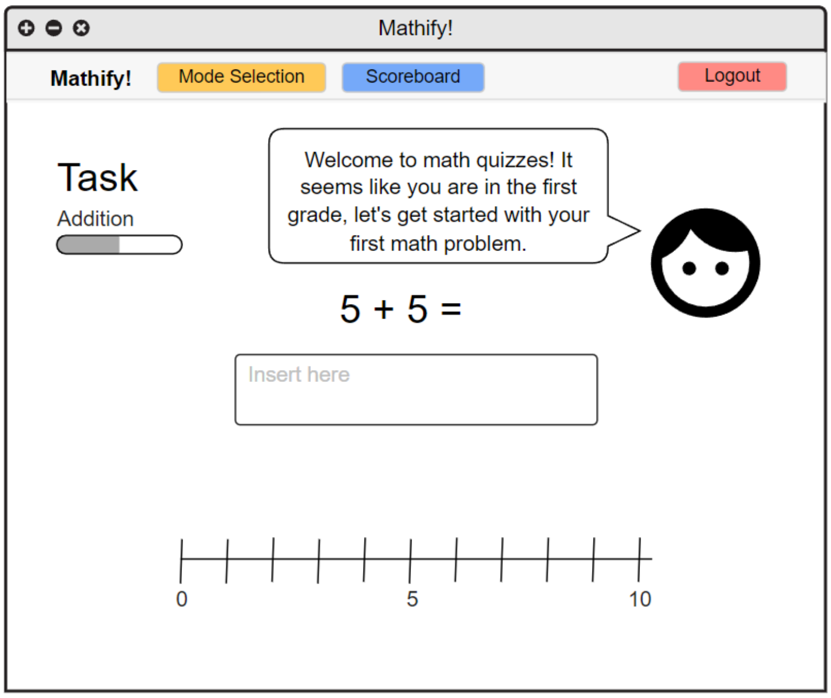

Mathify!
Mathify!
Mode Selection
Scoreboard
Logout
Task
Addition
Welcome to math quizzes! It seems like you are in the first grade, let's get started with your first math problem.
5 + 5 =
Insert here
0
5
10

Figure 5: Mock-up Math Quiz

Extensions / Alternate Flows:

<u>If the child solves an exercise incorrectly:</u>

1. The system notifies the child of the incorrect result.
2. The child is able to try again.
3. The child is able to display the correct result by pushing a button.

<u>If the child's answer is in the wrong format (e.g. a word instead of a number):</u>

1. The system generates an error message.
2. The system instructs the child on the correct format.
3. The child is prompted to re-enter their answer.

Special Requirements:

1. Text must be legible from 0.5 meters away.
2. The system must process validations within 2 seconds.
3. The level of difficulty[9] adjusts according to the correctness of the result.

[9] See Glossary for a definition of this term.

List of Technical and Data Variations:

- 2a) Input using a standard keyboard.
- 2b) Input using an on-screen keyboard in the browser.

Frequency of Occurrence:

- Whenever the child wishes to practice, which is the app's primary use case.

Open Questions:

- How many exercise types from the syllabus will the program incorporate?
- How to adjust the level of difficulty?

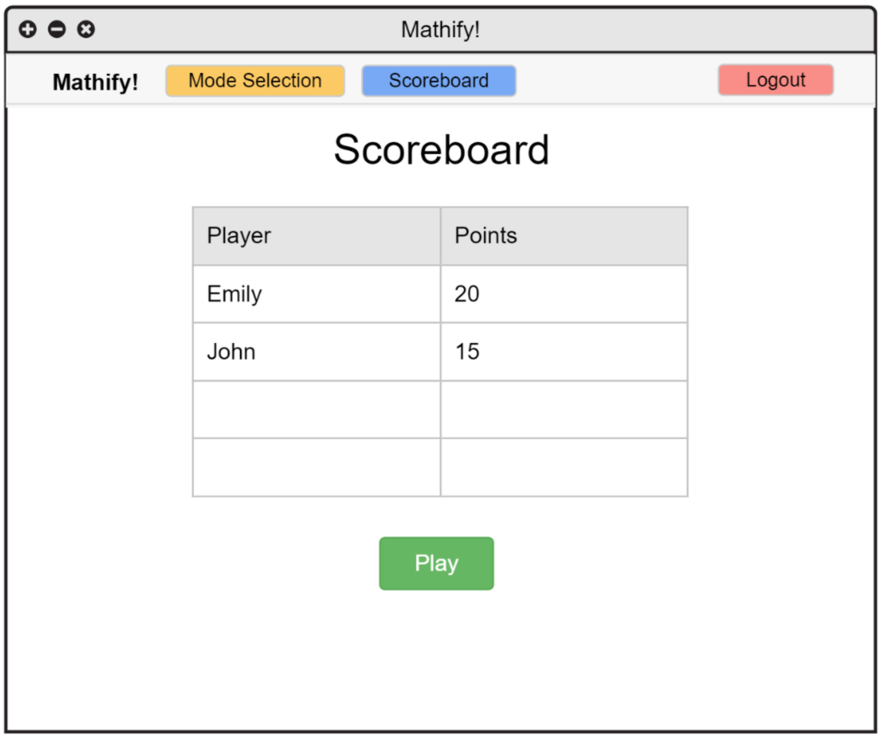
2.4.4 Gather additional Math Knowledge (Casual UC)[10]

- The child is able to access supplementary material related to each exercise type from a comprehensive teaching library[11] by pressing a button.
- Furthermore, when a low score is achieved on an exercise type, the system recommends additional learning materials automatically.
- Once the child feels confident with the topic, they return to the exercise screen to attempt further exercises.

2.4.5 Calculate / Display Score (Brief UC)

- The system updates the child's score, awarding ten points for each correct answer and one point for a wrong one. Following each exercise, the scoreboard[12] entry is updated.
- Upon reaching a specified threshold, the scoreboard is displayed.

[10] This use case is not part of the prototype.
[11] See Glossary for a definition of this term.
[12] See Glossary for a definition of this term.



Scoreboard

Player	Points
Emily	20
John	15

Figure 6: Mock-up Scoreboard

2.4.6 Adjust Level of Difficulty (Casual UC)

- When the child achieves a high score at a certain level of difficulty, a new level is unlocked, introducing more challenging exercises. The child can then choose to progress to the new level or opt to continue at the current level. After making their selection, the system reverts to exercise mode.
- Conversely, when the child scores low at a certain level of difficulty, the level is further lowered or remains the same if at the basic level already.

2.4.7 Create individual Exercises / Tutorials (Casual UC)[13]

- The teacher creates exercises or learning materials tailored to the class and uploads them to the app. The app checks the format and file size for compatibility.
- The teacher assigns the material to a specific topic and difficulty level or creates new topics altogether.
- The added content is then made available in the child's version of the app.

[13] This use case is not part of the prototype.

2.5 Step 5: Draw Use-Case-Diagram

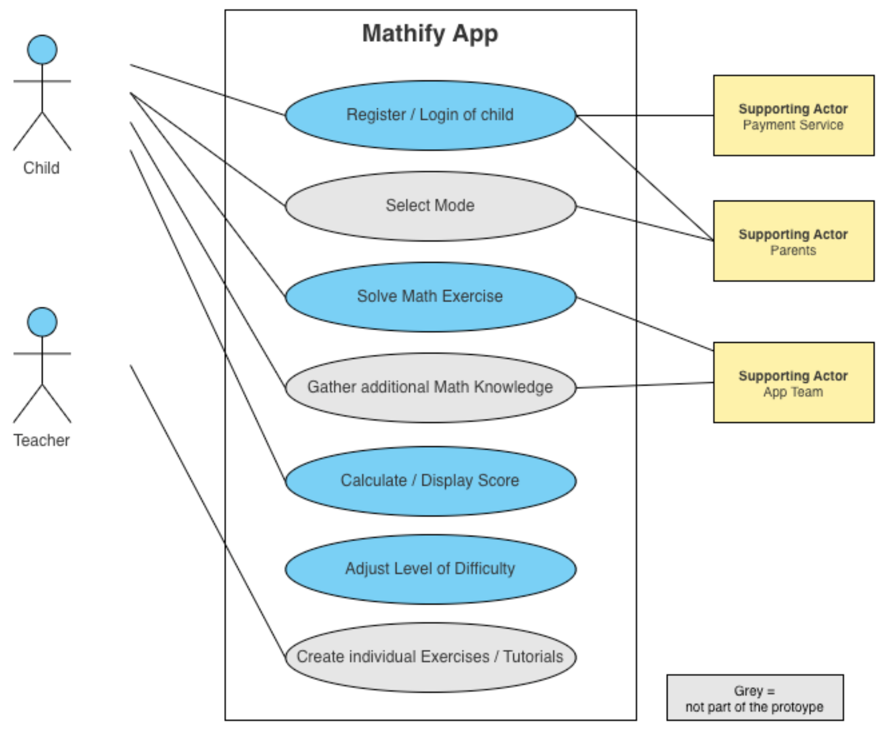

Mathify App
Child
Teacher
Register / Login of child
Select Mode
Solve Math Exercise
Gather additional Math Knowledge
Calculate / Display Score
Adjust Level of Difficulty
Create individual Exercises / Tutorials
Supporting Actor Payment Service
Supporting Actor Parents
Supporting Actor App Team
Grey = not part of the protoype

Figure 7: Use-Case Diagram

2.6 Step 6: Draw System Sequence Diagram (SSD)

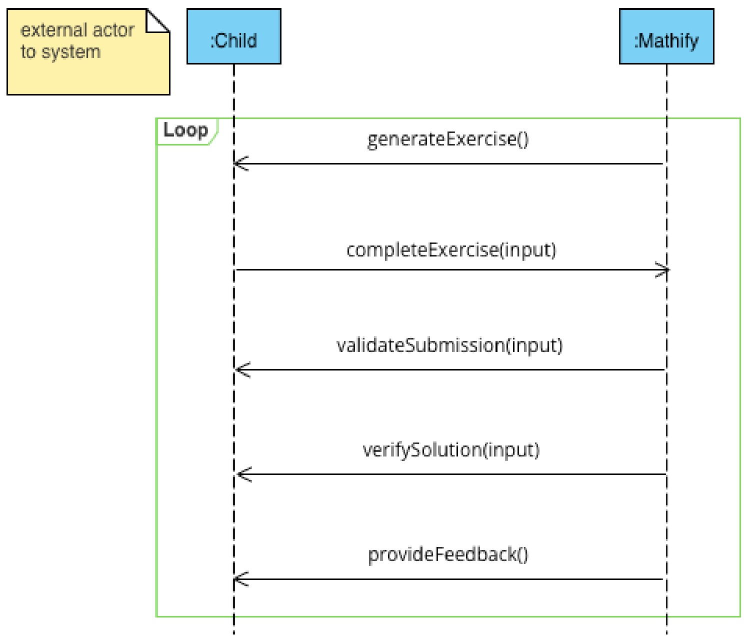


Figure 8: System Sequence Diagram of Main UC

3 Additional Requirements

The following chapter describes additional functional and non-functional requirements not covered in the use cases. The requirements are formulated as user stories and are all measurable / verifiable. Additionally, the team has limited itself to only stating requirements that are necessary without providing specific solutions on how to achieve them.

Our subject expert demands that...:

1. the child requires a maximum of three clicks after logging in to reach the first exercise, as exceeding this number of clicks may overwhelm cognitive capacity.
2. a screen should never contain more than two full sentences, and a sentence should never exceed ten words, as experience shows that excessive reading quickly diminishes the playful component.
3. results are saveable to track learning progress together.
4. the app should have no downtime between 08.00 and 18.00 on weekdays to enable proper integration into the curriculum.

Parents demand that...:

1. the payment method is universally accepted, as no new payment medium is opened solely for using an app.
2. the learning progress is visible as they are only willing to pay for useful tools.

Schools demand that...:

1. changes in the national syllabus[14] or new learning insights are continuously reflected in the app, as this is the only way a usage in the educational context makes sense.

[14] See Glossary for a definition of this term.

The government demands that…:

1. data security must always be ensured because sensitive data[15] is stored in the system, which makes security necessary from a regulatory point of view.

The company Mathify demands that…:

1. the app guarantees a traffic of 10,000 parallel users without breaking down to ensure credibility in the market and hence achieve profitability of the company.
2. The app must be compatible with common operating systems to quickly gain widespread adoption among schoolchildren and push sales figures up.

[15] See Glossary for a definition of this term.

4 Domain Model

The domain model[16] illustrated below represents Mathify through a simplified UML class diagram. It delineates the principal concepts with interconnection to each other to visualize their relationships. Attributes have been included where they enhance understanding.

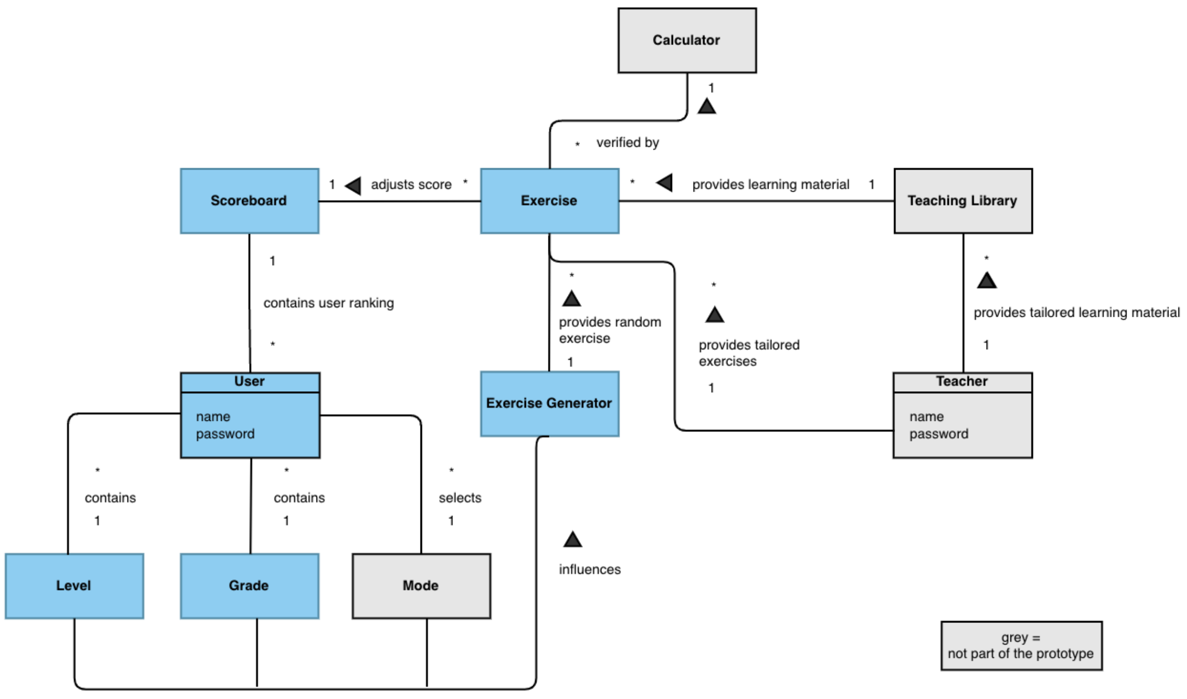


Figure 9: Domain Model

[16] See Glossary for a definition of the technical terms in the domain model.

5 Software Architecture

The decision on a specific software architecture and technology stack is a critical step in the development of a project, which has far-reaching implications for its success, maintainability, and scalability. In the case of Mathify, the decision was made based on several factors such as team experience, specific project requirements and past experiences with technologies.

5.1 Programming Languages

Java was chosen as the primary programming language, mainly because of the existing experience within the team. This decision allows the team to be immediately productive, as no additional learning time for a new language is required. Java is also known for its robustness, performance, and wide support through a variety of libraries and frameworks, making it a safe choice for our backend development.

Python[17] will be introduced later for specialized mathematical calculations. Python offers an extensive selection of scientific libraries like NumPy and SciPy, which are ideal for mathematical tasks. This decision enables us to leverage Python's capabilities in this area without having to build our entire architecture around it.

5.2 Frameworks

Javalin, a lightweight framework for the backend, was chosen to simplify development and maximize performance. Renowned for its simplicity and efficiency, Javalin enables backend development without the complexity and overhead associated with more extensive frameworks.

For the frontend, a combination of Angular, Bootstrap and Angular Material UI was selected. Angular allows for a modular and component-based architecture, promoting code reusability and a clear structuring of the app. Bootstrap and Angular Material UI complements Angular with simple and responsive design templates, accelerating the development of a professionally looking frontend.

5.3 Patterns and Architecture

The use of the Factory Pattern supports the creation of the Exercise Generator by providing a clear separation between the creation of objects and their use. This

[17] Python is not part of the prototype.

enhances the flexibility and extensibility of the code. Moreover, polymorphism is integrated through an 'Exercise' interface, which is implemented by various exercise types. This approach ensures that the system maintains high cohesion and low coupling by allowing the same interface to be used for different underlying types of exercises, which can be extended anytime without affecting other parts of the codebase.

The Model-View-Controller (MVC) pattern is employed to ensure a clear separation between the user interface, business logic, and data access layers. This separation facilitates ease of maintenance and scalability of the application.

The Client-Server architecture forms the basis of our system, enabling light and efficient communication between the frontend and backend, ensuring that the system can handle requests and scale as necessary without any significant changes to the client or server codebase.

5.4 Client-Server Architecture and Communication

In addition to the above-mentioned Client-Server architecture, it is important to detail the specific distribution model and communication strategy employed. Our application is a distributed system where the client (user's browser) interacts with our server through a web-based interface. The user accesses the website, which serves as the client, and makes requests to the server via an API. The server, built with our chosen tech stack, handles these requests in conjunction with user permissions, ensuring that users can only access the features and data they are authorized for. This is critical for maintaining security and data integrity across the system.

We opted for a RESTful API approach because it provides a standard and stateless communication protocol between the client and server, which simplifies scalability and the integration of future services. This design choice supports our distribution model by allowing the client and server to evolve independently if the API contract is maintained. Furthermore, we chose to implement a straightforward and minimalistic approach to our system architecture to minimize the amount of boilerplate code necessary. This not only allows for quicker development and easier understanding of the system's workings but also reduces the potential for bugs and simplifies maintenance and future enhancements.

5.5 Reusable Logic and Modular Structures

Outsourcing the calculation logic into a separate maths package allows for its reuse in distinct parts of the app, improving consistency and reducing the likelihood of errors.

The Exercise Generator is an example of a modular structure accessible through the Factory Pattern. This ensures that changes to the creation logic are centralized, simplifying the maintenance, and testing of the app.

5.6 Package Diagram

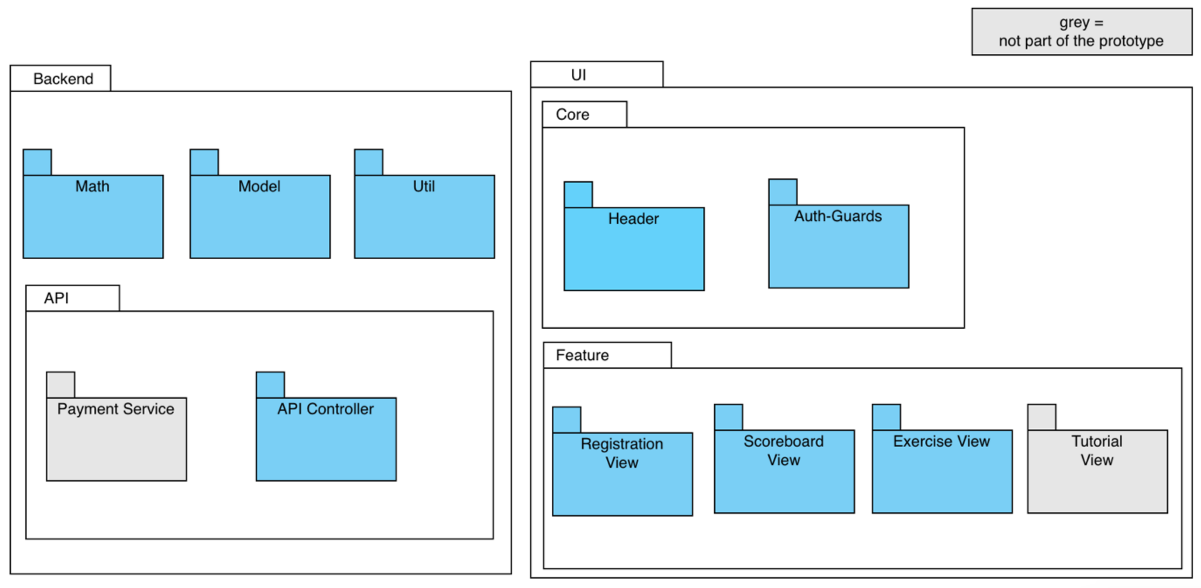


Figure 10: Package Diagram

5.7 Conclusion

The chosen architecture and technology stack reflect a careful consideration of the project requirements, existing team skills, and best practices in software development. These decisions enable us to develop a robust, maintainable, and scalable app that meets both current and future needs.

6 Design-Artefact

In this section, we present the design artifacts of the Mathify software, which serve as foundational building blocks for development and implementation. The design artifacts include the Design Class Diagram (DCD) and interaction diagrams for the system's key operations. These elements document the structural and interactive aspects of the Mathify application.

6.1 Design-Class Diagram (DCD)

The Design Class Diagram for the Mathify system displays the structural relationships and interactions between various components including User, Grade, Exercise, and additional elements. The User class handles user-related data and interactions, focusing on secure and personalized user experiences. The attributes and methods support user management, authentication, and experience customization.

The Exercise interface and its concrete implementations, such as MathBasicExercise, offer a range of mathematical challenges designed for users across different grade levels, reflecting the educational aspect of the system.

The Grade class serves as a cornerstone for monitoring user performance and progress. It interfaces with the Exercise classes to assess and record the outcomes of exercises completed by the users.

Classes like ExerciseGenerator and ExerciseSubType are instrumental in creating exercises that cater to the user's grade level and preferred types of math exercises, ensuring content is tailored to each user's learning curve.

The Router class is vital for managing API calls, delegating requests to the appropriate controllers, like UserApiController and ExerciseApiController. This central routing system is key to the seamless data flow and provides endpoints for the application's frontend to interact with the backend services.Supporting classes such as UserRepository and Settings are critical for user data persistence and application configuration, respectively. UserRepository is involved in storing and managing user information, whereas Settings handle application parameters, potentially including security features like HTTPS.

The diagram centers on these fundamental components, illustrating the data processing and representation mechanisms within Mathify, highlighting the system's design to ensure it remains an effective educational platform.

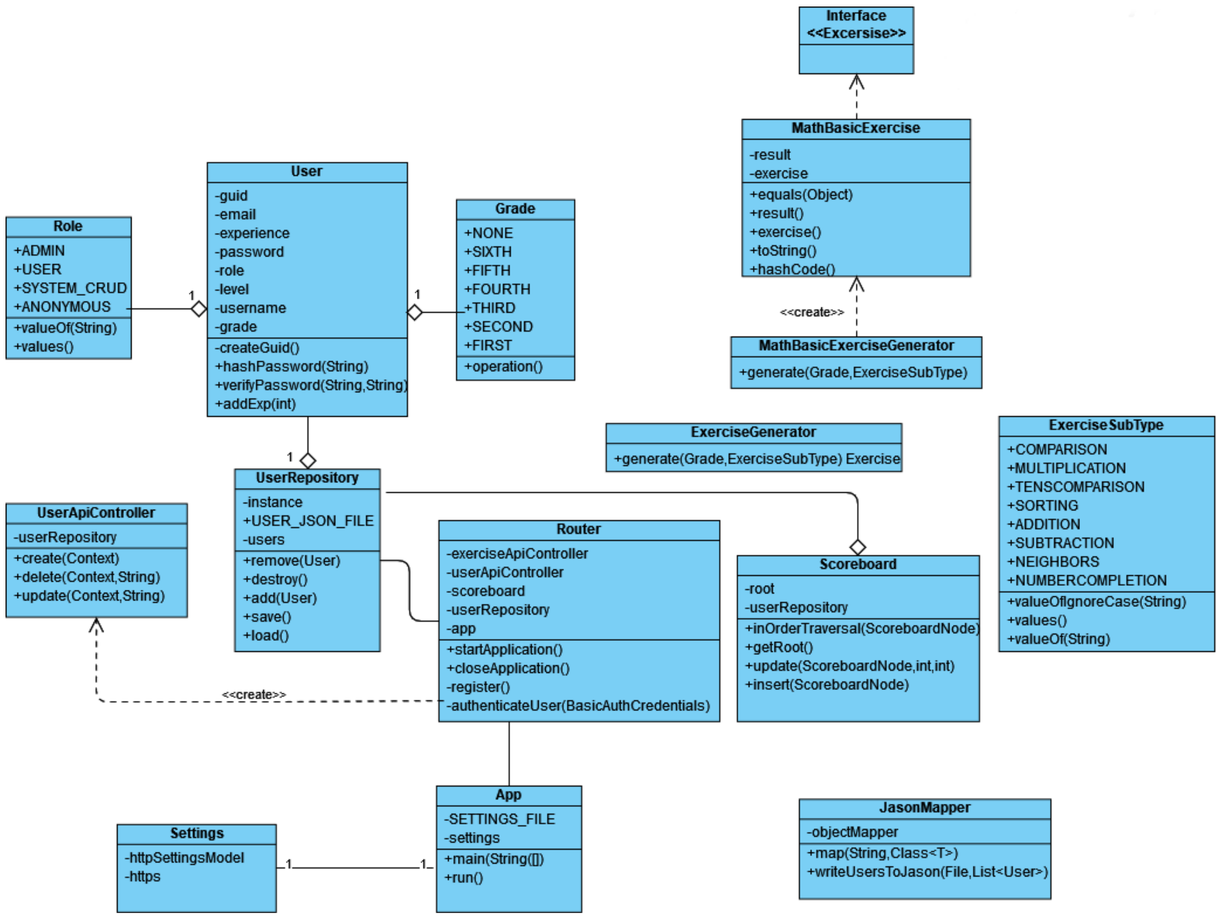


Figure 11: Design Class Diagram

6.2 Interaction Diagrams

The following section elucidates four key operations within the Mathify application, illustrated through interaction diagrams. These diagrams delineate the message flow between system components and showcase (sd in this context has no meaning):

Exercise Generation (Figure 11): The diagram illustrates a cyclical process in which students actively engage with the system to enhance their learning experience. Outside the 'sd loop,' students begin by selecting a game mode, a choice that is communicated to the System through the selectGameMode. This selection is crucial as it dictates the type of exercises the student will encounter, allowing for a tailored learning experience.

Upon receiving the student’s selection, the System then employs the generate() function to create an exercise that aligns with the chosen game mode. The generation of exercises is dynamic and responsive, ensuring that the educational content is both relevant and challenging based on the student's input.

Following the creation of the exercise, the displayExercise action is triggered, where the newly generated exercise is presented back to the student for interaction. This presentation is an integral part of the loop, as it keeps the student engaged in a continual learning and practicing sequence.

The 'sd loop' indicates that this is an iterative process, with the potential for students to select various game modes and receive corresponding exercises in a continuous loop, fostering an interactive and adaptive educational environment.

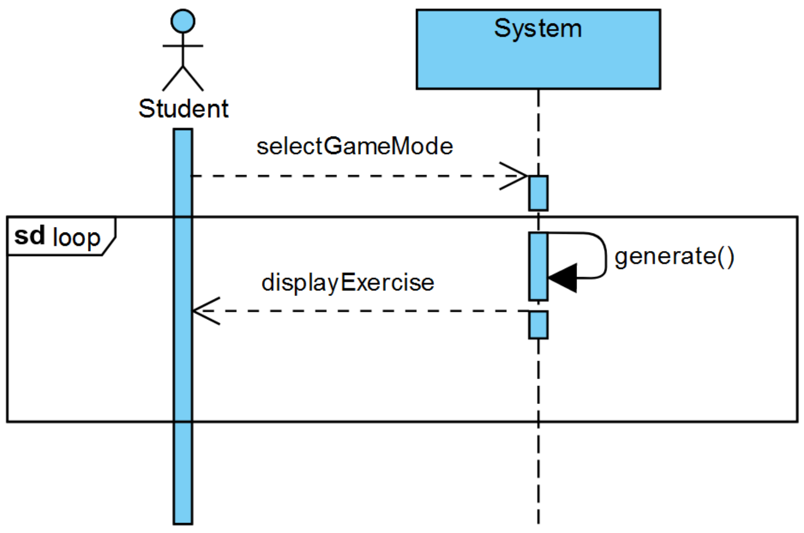


Figure 12: Interaction Diagram, Exercise Generation

User Registration (Figure 12): The sequence of user registration within the system is initiated when the startApplication() function is called, indicating that the system is ready to operate and interact with users. The process then moves forward with the provideEndpoints action, indicating that the frontend is equipped to handle further operations.

Subsequently, the user, through the frontend, initiates the register() function to submit their registration details. These details are sent to the API Controller, which invokes the validateUser() function to ensure the information meets the registration criteria. Upon successful validation, the create() function is executed, leading to the creation of the user's account.

The sequence concludes with the API Controller performing the providesUserState action, transmitting back to the frontend a complete set of user-related information. This information includes, but is not limited to, the user's grade level, points on the scoreboard, and other pertinent data. This wealth of information empowers the frontend to offer a personalized user experience, customized educational pathways, and access to system features aligned with the user's academic standing. This forms the basis for a tailored and insightful user journey within the system.

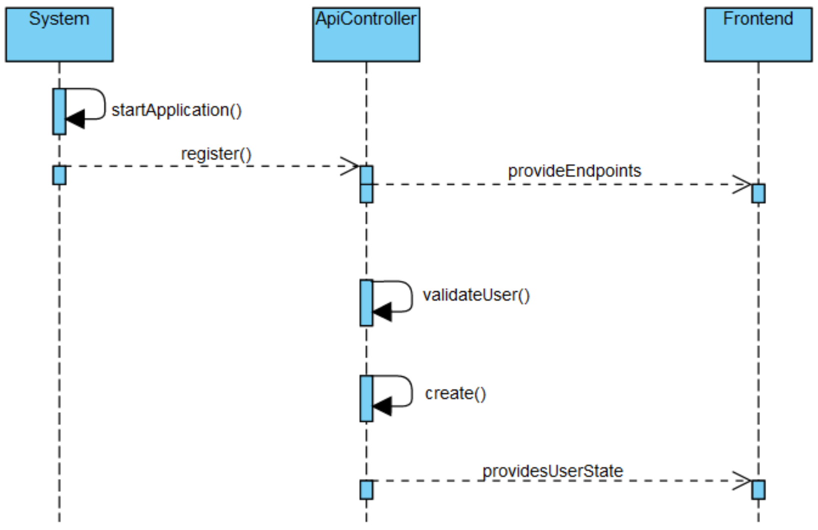


Figure 13: Interaction Diagram, User Registration

Score Calculation (Figure 13): As students interact with the system, they enter a loop of activity aimed at solving educational exercises. When a student attempts an exercise, the solveExercise interaction sends their response to the System for evaluation. The System's verifySolution function is then responsible for assessing the correctness of the student's answers in real-time.

Once a solution is verified, the updateScore action is executed, which adjusts the student's score based on the outcome of their attempt. This real-time score update is an essential aspect of the learning experience, providing immediate feedback on performance. The displayScore interaction ensures that the newly updated score is reflected on the student's interface, maintaining a clear and current view of their progress.

The continuous loop, denoted by 'sd loop,' represents the ongoing engagement of the student with the system, allowing for a consistent and iterative learning process where each new exercise solved contributes to an evolving score.

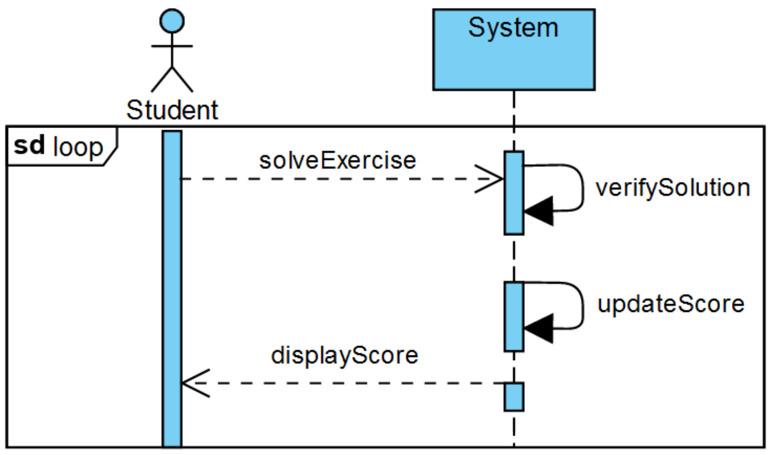


Figure 14: Interaction Diagram, Score Calculation

Difficulty Adjustment (Figure 14): This diagram demonstrates the interactive and adaptive learning process experienced by students within the system. Students engage in a series of exercises by solving the problems presented to them. Each solution attempt is analysed in real-time by the System, utilizing the verifySolution function to assess correctness. Upon successful verification, students are awarded experience points through the addExp() function, which contributes to their overall learning progression.

Subsequently, the System's adjustDifficulty mechanism tailors the complexity of subsequent exercises to match the student's evolving proficiency level. This adaptive approach ensures that the challenges are neither too easy nor too hard, maintaining an optimal learning curve. The loop continues as the System generates a new exercise appropriate to the adjusted difficulty level, presenting it to the student for the next attempt. This process encapsulates a tailored educational experience, fostering continuous engagement and development through personalized exercise difficulty.

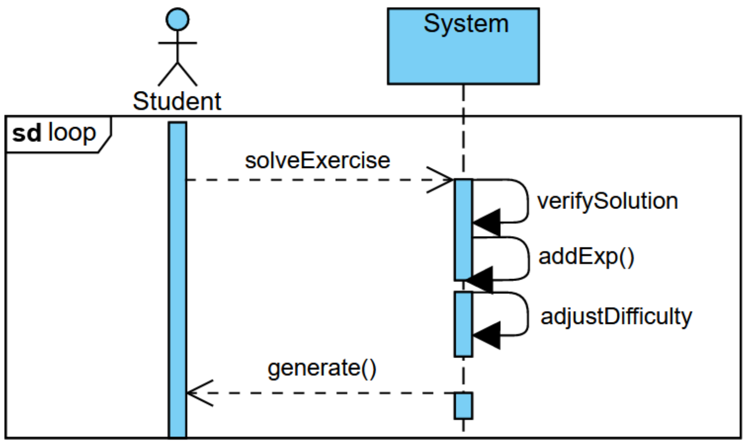


Figure 15: Interaction Diagram, Difficulty Adjustment

7 Implementation

A prototype representing the fundamental use cases is in the process of being developed to verify the selected software architecture. The construction of this prototype is essential for testing the viability and functionality of the architecture, providing a platform for identifying and implementing necessary adjustments at an early stage.

7.1 Backend Testing Strategy

The testing strategy for the backend is structured across various levels to ensure the system's robustness and integrity:

Unit Testing

The backends' unit testing is conducted to evaluate the functionality of individual methods, such as different exercises or access management functions. To go beyond traditional class-based testing, mock testing is extensively leveraged, allowing for simulation of interactions with external dependencies by mocking the relevant classes. This enhances test specificity and enables concentrated scrutiny of the units under test.

Integration Testing

Alongside unit tests, integration tests are carried out to verify the correct interaction between disparate classes. A case in point is examining the interplay between the User Repository, the User API Controller, and the User class. These tests are critical in ensuring seamless operations between components and in preserving the architectural integrity.

System Testing

Comprehensive system tests are performed to check the backend components' interactivity, primarily through manual testing. The goal is to ensure components such as the scoreboard, user list, and exercises are accurately orchestrated through the API. Such tests are indispensable for confirming the overall system's synchronicity and for replicating application behaviour under realistic operational conditions.

7.2 Frontend Testing Strategy

In the realm of frontend development, a conscious decision was made to forego unit testing in favour of manual testing strategies. This decision was informed by a thorough evaluation of the cost-benefit ratio, wherein the effort required to establish a framework for unit tests did not align with the value they were expected to provide. Manual testing is given precedence in the frontend to effectively uncover issues that affect the user experience, which are often more nuanced and subjective than those detected by automated tests.

8 Project Management

Effective project management is crucial for the success of software development projects. This chapter examines the approach to project management to develop Mathify. It includes reviews of planned tasks versus actual outcomes, along with necessary adjustments. Additionally, there is a revisited risk assessment and an outline of proactive planning for upcoming sprints. The focus is on detailing the intricacies involved in managing the Mathify project.

8.1 Sprint 1

The initial sprint of our Mathify project was dedicated to establishing the foundational elements of our application. The sprint's scope, along with the planned and actual efforts, is detailed in Table 1.

Task	Planned Effort	Actual Effort	Result
Establishing the initial project structure	S	S	Completed
Modelling user and exercises	M	M	Completed
Setting up back- and frontend	M	L	Partial Completion
Building a backend testing environment	M	M	Completed
Formulating a skill table as a basis for generating exercises	S	M	Completed
Compiling a readme document	S	S	Completed

Table 1: Sprint 1 Review

The primary objective not realized within this sprint was the implementation of backend authentication, which has been postponed to Sprint 2. The deferral is due to the task requiring a greater effort than initially anticipated, coupled with the team's limited experience in this domain. Furthermore, the effort estimate for the 'Skill Table' was adjusted from S to M, reflecting the underappreciation of the complexity involved in comprehending the national syllabus, 'Lehrplan 21,' and identifying the most effective exercise types for each grade to optimize educational outcomes for children. Despite these setbacks, the project's schedule buffer has been considered adequate to absorb these delays, and therefore, no immediate corrective actions have been initiated.

8.2 Sprint 2

In our second sprint, we addressed the deferred tasks and introduced new functionalities. The sprint's objectives and their progress are outlined in Table 2.

Task	Planned Effort	Actual Effort	Result
Backend authentication	M	M	Completed
GUIs for user registration, grade selection and mode selection	M	L	Registration Window Pending
Test frontend to backend connection	M	L	Pending Implementation
Good draft for technical report 1	M	M	In Progress
Implement grade 1 exercises	S	M	Completed
Implement scoreboard	L	S	Completed

Table 2: Sprint 2 Review

The GUI for user registration, along with grade and mode selection, is pending a separate registration window, and the connection between the front- and backend is yet to be established. Despite these unfinished items, the team's morale and confidence remain high, with the consensus being that the final goals will be achieved without additional measures. The project remains on track, with no further measures required as team collaboration continues to be strong.

8.3 Sprint 3

Sprint 3 has concluded, demonstrating the team's ability to accomplish set goals within a condensed timeline of just one week instead of two. The sprint's deliverables, with their planned efforts, are summarized below:

Task	Planned Effort	Actual Effort	Result
Connect front- and backend	M	L	Completed
Login page	M	M	Completed
Refactor exercise generation	S	S	Completed
Implement grade 2 exercises	S	S	Completed
Create router mock tests and improve existing ones	M	M	Completed
Create window for registration	S	S	Completed
Scoreboard and exercise presentation on frontend	M	M	Partial Completion
Finalise technical report 1	M	L	Completed

Table 3: Sprint 3 Review

The completion of Sprint 3 within its ambitious timeline highlights the team's dedication and the effectiveness of our project management methodologies.

8.4 Sprint 4 and 5

The plans for sprints 4 and 5 are outlined in the two tables below. Please note that these are still preliminary estimates.

Task	Planned Effort
Implement grade 3 & 4 exercises	S
Implement user persistency after shutting down the app	M
Implement technical score	M
Overhaul exercise numerical value logic	S
Good draft of technical report 2	L
Enhance user experiences for exercises	L
Implement comparison, sort, etc.	M
Finalize implementation of scoreboard in frontend	M

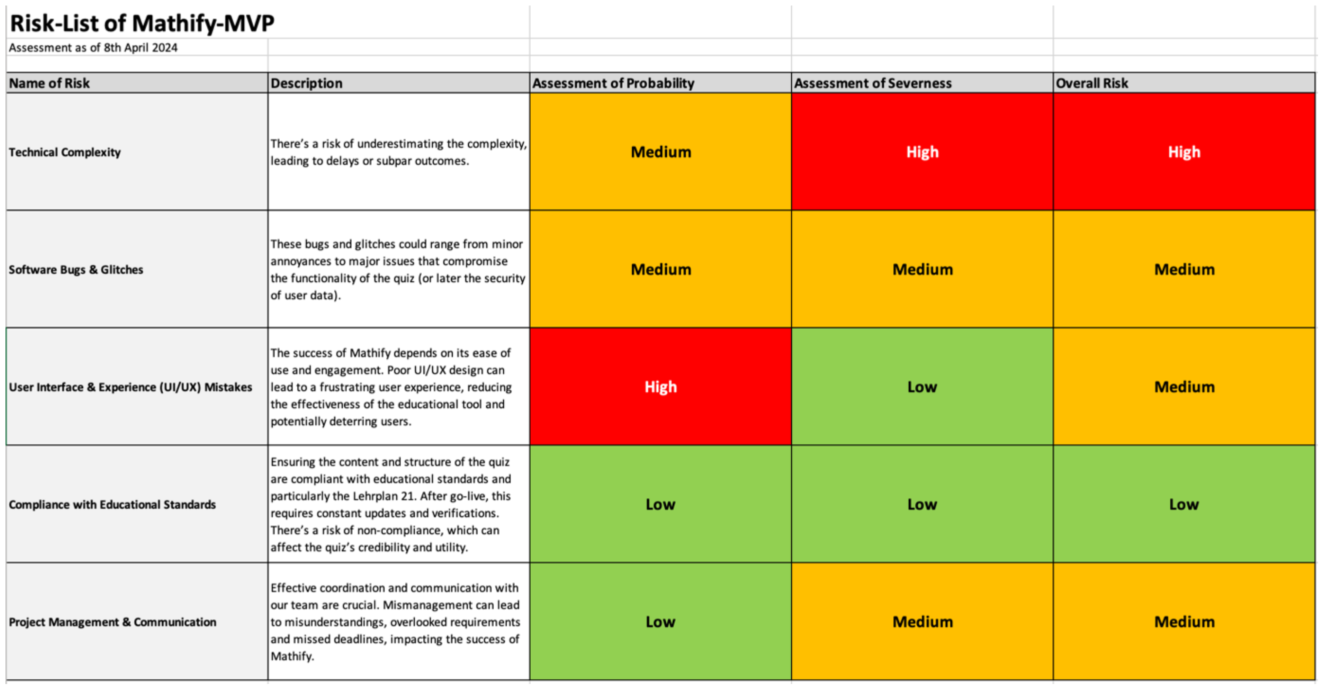
Table 4: Sprint 4 Planning

Task	Planned Effort
Implement grade 5 & 6 exercises	S
Bugfixing	-
Finalizing technical report 2	L

Table 5: Sprint 5 Planning

8.5 Updated Risk Assessment

The risk assessment is regularly revised to account for the dynamic project environment. The team has effectively addressed numerous risks and alleviated potential problems. A comprehensive risk assessment is provided in the figure below:



Risk-List of Mathify-MVP

Assessment as of 8th April 2024

Name of Risk	Description	Assessment of Probability	Assessment of Severness	Overall Risk
Technical Complexity	There's a risk of underestimating the complexity, leading to delays or subpar outcomes.	Medium	High	High
Software Bugs & Glitches	These bugs and glitches could range from minor annoyances to major issues that compromise the functionality of the quiz (or later the security of user data).	Medium	Medium	Medium
User Interface & Experience (UI/UX) Mistakes	The success of Mathify depends on its ease of use and engagement. Poor UI/UX design can lead to a frustrating user experience, reducing the effectiveness of the educational tool and potentially deterring users.	High	Low	Medium
Compliance with Educational Standards	Ensuring the content and structure of the quiz are compliant with educational standards and particularly the Lehrplan 21. After go-live, this requires constant updates and verifications. There's a risk of non-compliance, which can affect the quiz's credibility and utility.	Low	Low	Low
Project Management & Communication	Effective coordination and communication with our team are crucial. Mismanagement can lead to misunderstandings, overlooked requirements and missed deadlines, impacting the success of Mathify.	Low	Medium	Medium

Figure 16: Risk-List

8.6 Conclusion

The completion of Sprint 3 represents a steady continuation of the Mathify project, aligning with the established project management plan. The team's ability to meet almost all scheduled targets in a shortened sprint period has been notable. While the project has progressed smoothly, with no significant issues thus far, this success is understood in the context of careful preparation and adherence to best practices rather than mere circumstance. This steady progress instils a quiet confidence in the team's future performance. The proactive identification of potential issues and the efficiency in addressing smaller, manageable problems has contributed to a positive project outlook. However, the team remains aware of the complexities ahead and is prepared to approach future tasks with the same diligence and attention to detail that has characterized the project to date.

The project moves forward on solid footing, with each phase building on the lessons learned from the last. This incremental progress—free from major disruptions—affirms the robustness of our project management strategies and the resilience of the team. As we continue, we maintain a cautious optimism, conscious of the challenges that lie ahead but equipped to handle them with the competence demonstrated thus far.

9 Glossary

Term	Definition / Explanation
App-Team	The collective responsible for the development and maintenance of the Mathify quiz platform.
Calculator	A component within the app designed to assess and validate the accuracy of answers submitted by the user.
Child	The primary user group for the Mathify quiz, encompassing students from grades 1 through 9.
Exercise	A mathematical challenge tailored to the student’s grade, selected mode, and reached level of difficulty.
Exercise Generator	A system component that creates a variety of random mathematical exercises for users.
Game Mode	An initial selection feature that allows users to choose a play style such as mixed or custom.
Grade	A categorization that ranges from 1 to 9, dictating the types of exercises generated based on educational level.
Teaching Library	A repository of educational resources, including videos, texts, and graphs to support learning.
Level of Difficulty	A progression parameter within the game that progresses from basic to intermediate and finally to advanced.
Mathify	The official name of the quiz app.
National Syllabus	The “Lehrplan 21” dictates the variety and complexity of exercises tailored to each grade and level of difficulty.
Scoreboard	A feature that enables students to compare their performance with peers and their learning journey.
Sensitive Data	All personal information that can identify clients, requiring secure storage to protect privacy.

10 List of Illustrations

10.1 List of Figures

10.2 List of Tables