B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Exper - Gamified learning Environment

By
Shane Walsh

April 27, 2025

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.0.1 Project Context

My final year project is a Gamified Learning Environment. The problem I aim to tackle with this project is this: Studying techniques outside of class are largely not engaging and, for many students, fail to be effective and leave a lasting impression upon their learning. My personalised learning platform aims to remedy this. It's designed to engage students through interactive customisable quizzes and keep them studying long term through gamification and statistics tracking.

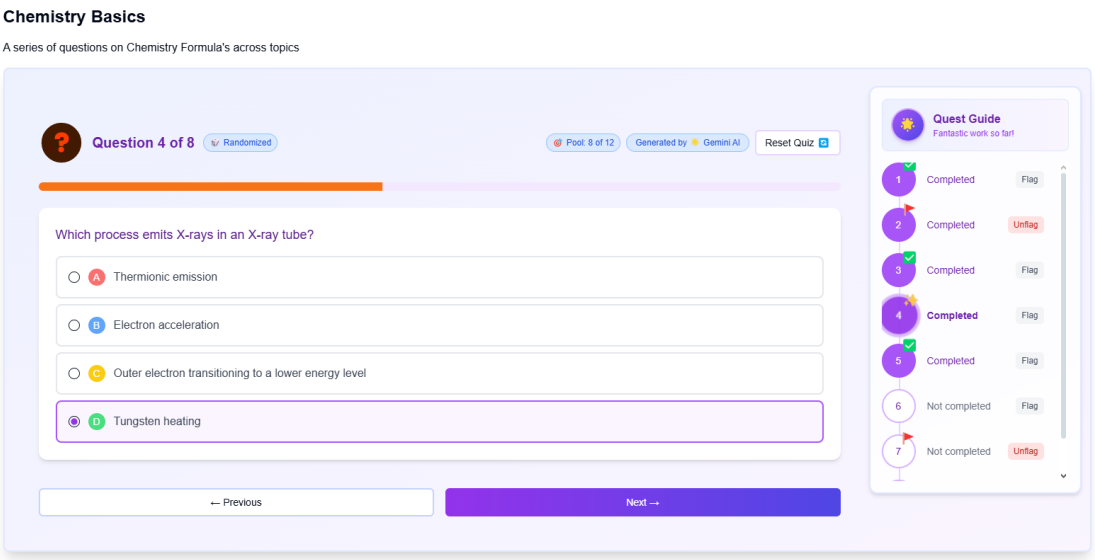


Figure 1.1: A Quiz Attempt in Exper

It leverages both pre-built data and generative AI to allow users to create quizzes for studying. A user can directly create a quiz crafted from their own notes (or even a teacher's), catering to their specific study needs at that moment. The gamification present throughout the app in a multitude of facets helps to keep students learning for longer, rather than put down their notebook for a "quick" 1 hour break. This gamification is complemented by progress tracking to provide users with statistics and feedback as they study, allowing them to get a return on their study that they typically lack. A microservice based architecture will allow for modularity within the codebase as this platform is developed and ensures a separation of concerns.

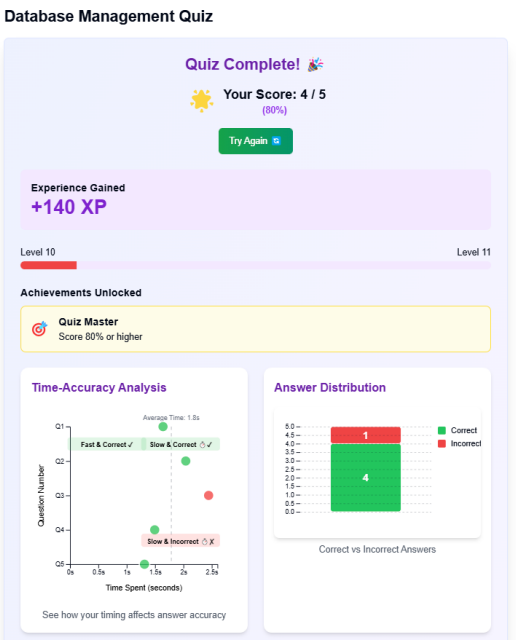


Figure 1.2: Quiz completion

1.0.2 Project Objectives

This project aims to develop a comprehensive Gamified Learning environment over a multitude of objectives:

Designing a personalised learning platform

The platform aims to feature interactive and customisable quizzes that can be adapted to an individual student's needs, their progress and their preferences. It should support multiple question types, levels of difficulty, and subject areas or learning categories. This ensures a broad appeal that makes it usable by students with many needs.

Robust Gamification Framework

Following suit from pre-existing Gamified Learning Platforms, the project aims to incorporate a range of features frequently seen in video games to keep the user motivated as they study. Features like achievements, badges, levels, progress tracking, leader boards, profile rewards etc. The systems present here will be designed with established motivational theories and best practices (in regards to gamification) in mind, based on undergone research into the topic to guarantee it is effective at incentivising the users.

Creating a Quiz Generation System using pre-built data and generative AI with prompt engineering

Quizzes are built using either pre-built data repositories and through the use of Generative Artificial Intelligence with prompt engineering. The system will be able to produce custom quizzes with quality questions. The questions generated should maintain pedagogical value (the theory and practice of teaching, adheres to teaching practices). Values for quiz generation should be tweak-able in a variety of ways, granting the user freedom in quiz creation.

Building an intuitive and responsive frontend

The frontend will provide a seamless intuitive experience across desktop and mobile. Throughout development, accessibility, clarity and ease of use will be prioritised. This will largely be done by leveraging Next.Js and Tailwind CSS to provide a seamless experience across devices.

Implementing a microservice architecture for modularity and scalability

In an effort to ensure modularity and maintainability within the platform, a microservice architecture will be implemented. This separation of concerns across multiple microservices such as User Management, Quiz Generation, Results Tracking, and Gamification will give the project an extendable nature, allowing for independent scaling of each element.

Tracking and providing dynamic feedback

Provide comprehensive tracking of user's results as they attempt quizzes and generate a variety of graphs from this. Allowing the user to identify where they are failing, their learning patterns, and more.

1.0.3 Technical Approach

The frontend will be developed using Next.js, the react framework. Its server-side rendering and efficient data handling, coupled with Tailwind CSS and Shadcn components, come together to help build a responsive design that will be both intuitive and visually pleasing. Data visualisation will be present throughout the app using D3.js, allowing for meaningful study feedback. Database storage is done with MongoDB, taking a NoSQL document-based approach.

This Gamified Learning Environment will be crafted modularly to ensure it

has a scalable and maintainable codebase. The system comprises of four main microservices.

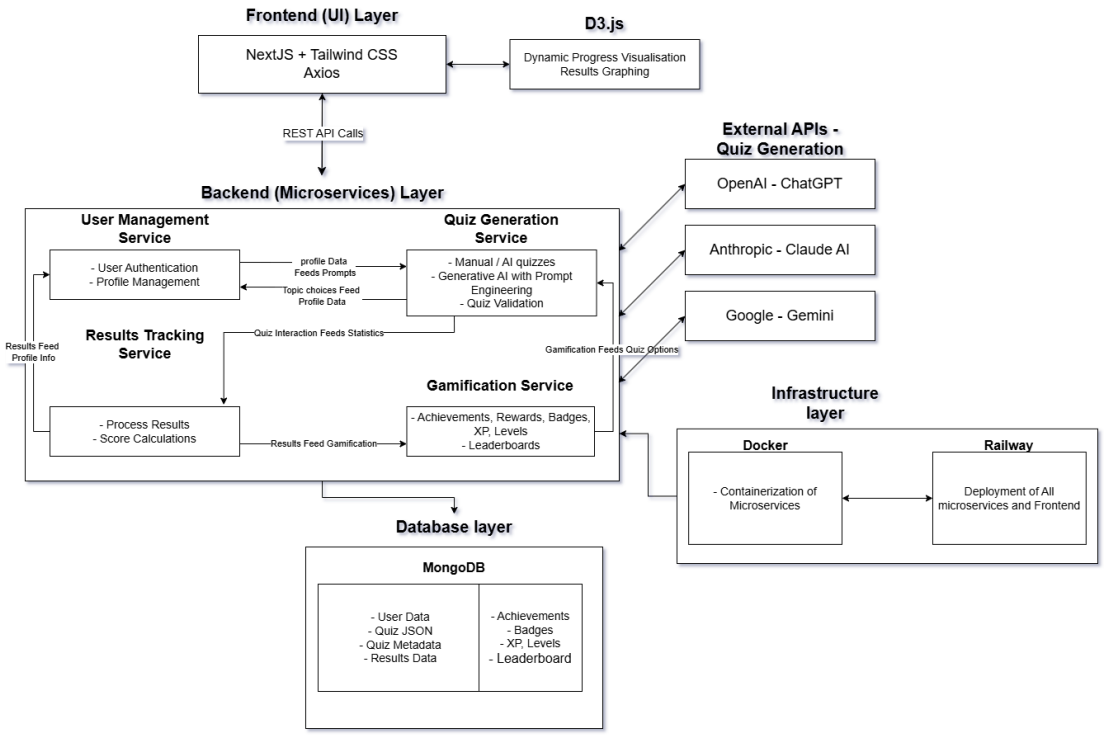


Figure 1.3: GLE Architecture Diagram

Microservice	Description
User Management Service	This handles user authentication and authorization, adhering to security norms and expectations. It also oversees user profile creation and management.
Quiz Generation Service	Manages all factors of the core quiz creation process. Providing both manual quiz creation and AI-powered quiz generation with integration between OpenAI's ChatGPT and Anthropic's Claude AI.
Results Tracking Service	Records performance and results of quizzes attempted, calculates scores and maintains long term performance data.
Gamification Service	Manages the various gamified elements such as achievements, badges, levels, category levels, leaderboards etc. Also manages the "Player" side of a user, aka the storage of a user's game stats.

Table 1.1: Microservices Architecture

1.0.4 Dissertation Structure

This dissertation is organised into the following chapters, providing a detailed look into each aspect of its development.

- Introduction - Establishes the project context, objectives, and relevance.
- Methodology - Details the development approach, including software development methodologies, planning processes, and research strategies.
- Technology Review - Evaluates the technologies selected for implementation and justifies these choices against alternatives.
- Design - Presents the system architecture, component interactions, and data flow within the GLE.
- Evaluation - Assesses the system against the established objectives and discusses its limitations and potential improvements.
- Conclusion - Summarizes the project outcomes and explores opportunities for future work.

1.0.5 Repositories

Exper is stored within a GitHub Organisation. This consists of multiple repositories, each managing a different aspect of the overall system. There is one repository for each microservice, keeping them separated and modular as intended. Then there is a repository containing the Next.js Frontend and one additional one for Deployment should it prove necessary. This encourages the modularity of the codebase set out in the objectives and ensures separation of concerns. From beginning to end, there was a total of 140 commits on the Frontend, and anywhere from 16 to 50 sizeable commits across the microservices.

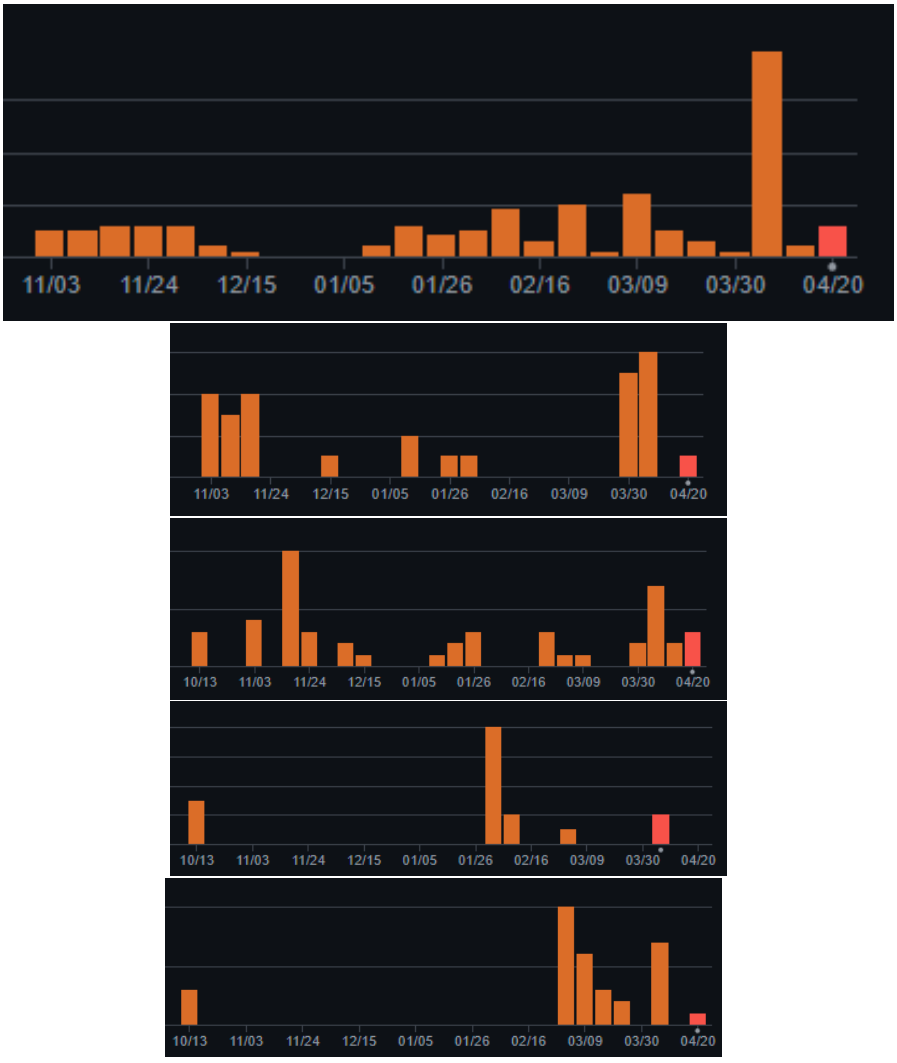


Figure 1.4: GitHub Commits throughout the Project in the following order: Frontend, User Management, Quiz Generation, Results Tracking, Gamification

A Comment on Relevance

Exper, the GLE developed within this project is addressing a serious and critical gap within modern education, in particular the studying and learning cycle. The lack of engaging, personalised motivational study tools for use outside the classroom. Education across secondary and tertiary levels (or even above) is moving increasingly online, relying on platforms like Microsoft Teams, Google Classroom, Moodle or in-house software to deliver learning material to students. There is a real growing need for solutions that allow students to study effectively outside of the classroom, with deep engagement and providing meaningful feedback alongside. [9] [10]
By combining interactive quiz functionality with personalisation powered by AI, gamification features and performance analytics. This GLE aims to significantly increase or transform educational technology, changing how we study on a daily basis. It adapts studying from a tedious task into an active experience that not only rewards but also is adaptable to a student's individual needs.

1.0.6 Inspiration

Much of the inspiration for this came from personal studying experiences throughout life and the post-pandemic shift-up of the educational landscape. Hybrid learning models have become more mainstream ever since Covid-19. Students should have effective tools to support this more independent study. The GLE's focus on engagement, motivation, and personalization directly addresses these evolving educational needs and demonstrates how technology can be used to enhance learning.

Chapter 2

Methodology

2.0.1 Development Approach

This project underwent an Agile development approach or methodology. More specifically, it adhered to the Scrum framework, this ensured flexibility and adaptability throughout its development and allowed for the separation of work into feasibly sized sprints. Gamified Learning and Artificial Intelligence are both fields that are evolving in how they are delivered. To be successful, they require frequent iterations and quick reactions to feedback given. Taking this into account made Agile the obvious choice.

Jira was utilised throughout the project as the primary project management tool. This was used to plan in advance, build out the project requirements backlog and separate them into sprints. This included the development and management of each of the microservices.

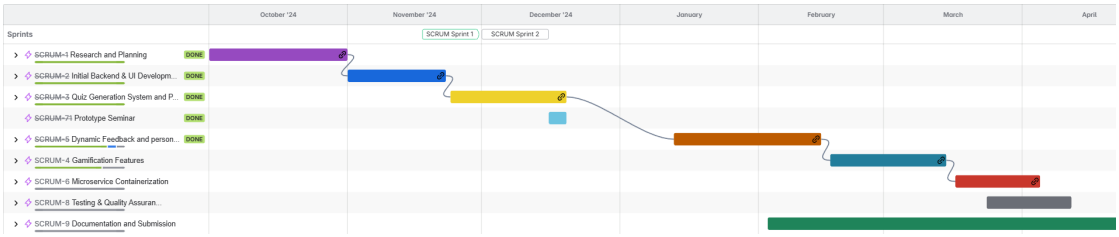


Figure 2.1: Jira timeline for Exper

The development was organised into sprints of various sizes depending on the workload at hand. Each sprint had specified deliverables and objectives in mind, aligning with the project's overall production. Each sprint was planned in advance and adhered to throughout the development process, with minor due dates pushed

forward or delayed to account for time needed for other modules. Feature delivery was largely prioritised, selecting them from the product backlog and breaking them down into manageable tasks.

Initially, GitHub Projects was utilised as the tool of choice for development planning and tracking of progress, leveraging its ability to directly connect the project's Git repositories to project boards, backlogs and more. This was severely early in the lifecycle, lasting from roughly early September to mid October in its use. This early phase saw much of the initial project requirements identified and allowed for the beginnings of a backlog of features to develop.

However, as the project developed, so did its needs. It became apparent that the project required a more intuitive and robust form of project management. While GitHub Projects benefited from having direct connection with repositories, it was found to be lacking in many other regards, particularly in its ease of use. Due to this, the project quickly pivoted to the use of Jira. Jira generally allowed for more advanced planning and detailed building out of requirements. In particular, it allowed for the distinct separation of each microservice's development into individual sprints.

Agile as a methodology was of great benefit for the following reasons:

- It's adaptiveness to changing requirements throughout the development process.
- Ability to deliver software components and features continuously and earlier in development.
- Convenient for integrating supervisor feedback, of which proved crucial throughout the project.
- Proved convenient for incremental development of the project's most complex elements, such as the Quiz Generation System and Gamification elements implementation.
- Effective with ongoing research, allowing continuous additions or changes to be made as the knowledge base grew.

2.0.2 Planning and determining Requirements

Determining requirements was an involved process filled with several aspects. Meetings with the supervisor quickly aided in identifying what the project's objectives were. It became quickly apparent, that it could not be simply a study

tool with some simple AI integration. It was important that where possible, the platform provided its own unique service that could not be found elsewhere. From a user perspective, this brought the project into two key overarching objectives:

- To utilise AI through prompt engineering to provide a highly customisable study environment, where users can study from their own personal materials.
- To integrate gamification, results tracking and an adaptable nature into every aspect of the platform to provide users with a form of study that is far more engaging and with much greater return for them when they study.

These high level goals called for the technological requirement of a micro service based architecture. One that would complement the malleable adaptable nature of the platform and allow for separating the development into more specific feasible portions. For example, the implementation of interactable quizzes that could be tailored to the individual needs of a student and the integration of several generative AIs for quiz generation (limiting the platform to one AI not only would've limited the scope of the project and reduce the opportunity for comparisons, but also would've made it reliant on it). Lastly the incorporation of gamification elements all over like progress tracking, achievements, levelling and leaderboards.

2.0.3 Supervisor Meetings

Consistent meetings with the project's assigned supervisor were crucial to its completion. They ensured continuous progress and kept the project aligned with the academic requirements. Meetings were instrumental in keeping the project accountable to its deadlines and schedules, while avoiding going overboard with features or lazing in effort.

Much went into the meetings but to summarize:
Weekly to Biweekly meetings with the assigned academic supervisor were scheduled in advance of their date and were kept to throughout the lifecycle of the project. Each session typically lasted at least 30 minutes, but typically went up to about 45 minutes to an hour.

Topics covered would include:

- Updates on progress made or features completed since the previous meeting, including explanations of their implementation.
- Discussions over roadblocks encountered or technical challenges and what solutions could be used to overcome them, or whether a new direction was necessary.

- Review of features completed and demonstration of their implementation in live demos, followed by discussion on their technical implementation, further additions, user interface/usability or additional complementary features.
- Feedback on the project's academic aspects, including research conducted and ongoing dissertation writing.
- Planning for what came next, setting in stone the future deliverables and developing an accountability to the deadlines set.

Overall, meetings with the supervisor were invaluable throughout the whole of the project's development, they helped with technical roadblocks, long-term discussions of implementations, academic questions involving dissertation development and maintain academic evidence while delivering technical features and functionality.

2.0.4 Key Discussion Topics from Supervisor Meetings

Chief talking points during these meetings influenced much of the planning process:

Key Discussion Topics

Prompt Engineering: Exploring strategies for effectively utilising prompt engineering for ensuring that quizzes generated are adaptable to parameters like difficulty and catered to a student's inputted notes.

User Experience: Defining user stories to better understand who this project's target market is and the expectations of them. This in turn formed how features were prioritised.

Data Management: Storage and retrieval planning of quiz data and results, user information or gamified statistics.

Microservices Architecture: Separating responsibilities of aspects of the project and defining them into individual microservices within the system architecture. This would ensure a scalable and maintainable platform not only during development but in eventual deployment too.

Feature-First Development: Working in a feature-first development format. This is where development of features is prioritised based on how it'll influence or enhance a user's engagement or use of the platform and how it'll meet the overall objectives of the project.

Gamification Strategy: Strategising the implementation of gamification elements to boost motivation to learn and engagement with the platform's systems. This involved a number of research into other gamified quiz platforms.

Deployment Planning: Considerations into deployment pipelines and the deployment process. Considering cloud based hosting like Digital Ocean, Railway, Vercel and more.

2.0.5 Development Tools

Selecting effective tools for development was invaluable to the project. This includes its development across each aspect and the deployment of the final Gamified Learning Environment.

Version Control

GitHub Organizations was utilized as the primary platform for version control. This provided numerous benefits, including:

- Separation of concerns, each microservice Backend was developed within its own repository, with another for the Frontend. This allowed for easier management of complexities in the codebases.
- Prior to moving to Jira's more robust systems, GitHub project's was utilized for the project's development planning. This made use of its project boards, backlog and GitHub's project visualization.
- Automated workflows were an option through the use of GitHub Actions.

Containerization and Orchestration

Docker was employed to allow for consistency across development, testing and production environments. They allowed for effective containerization of the various microservices.

- Docker containers were used for encapsulating each microservice, including its dependencies.

- Use of Docker compose facilitated local development and testing of the system when it was complete.

There were strong aspirations to use Kubernetes for its container management and orchestration capabilities. But this ultimately fell out of scope.

Development Environment

Visual Studio Code was the primary IDE for the whole of the GLE's development. It was configured with a suite of extensions to allow Python and Next.JS development, including integration extensions for Docker and Kubernetes.

This allowed development to be streamlined from writing code to deployment and dissertation writing, supporting the Agile methodology and workflow. Overall it reduced friction in much of the development process.

2.0.6 Additional Research

Gamification in Education and Engagement

Gamification is typically recognised as the applying of game design techniques and features to non-game situations. In the case of this project, it is applied to teaching and learning environments with the overall aim of increasing engagement. [11] Common elements seen include points systems, badges, leaderboards and narratives. These are used to enhance the experience of learning and increase interactivity. [12] Studies show that when these are incorporated, they have the chance to significantly increase student motivation and engagement. [13], which can in hand lead to increased participation in learning activities and discussions. [9] In general, research shows a positive impact from the implementation of these features to education. [5]. However its not always so simple, when looking at academic performance with gamified features at play, the results are mixed. It depends heavily on the implementation and how careful you are about it. Design of these features must keep core learning goals in mind, and not just get distracted in the bells and whistles of gamification. [14]

When designed carefully with this in mind, gamification has been shown to have positive impact in increasing academic achievement and student enjoyment in a subject. [1] It can studying from a tedious task (perception wise) to a rewarding experience.

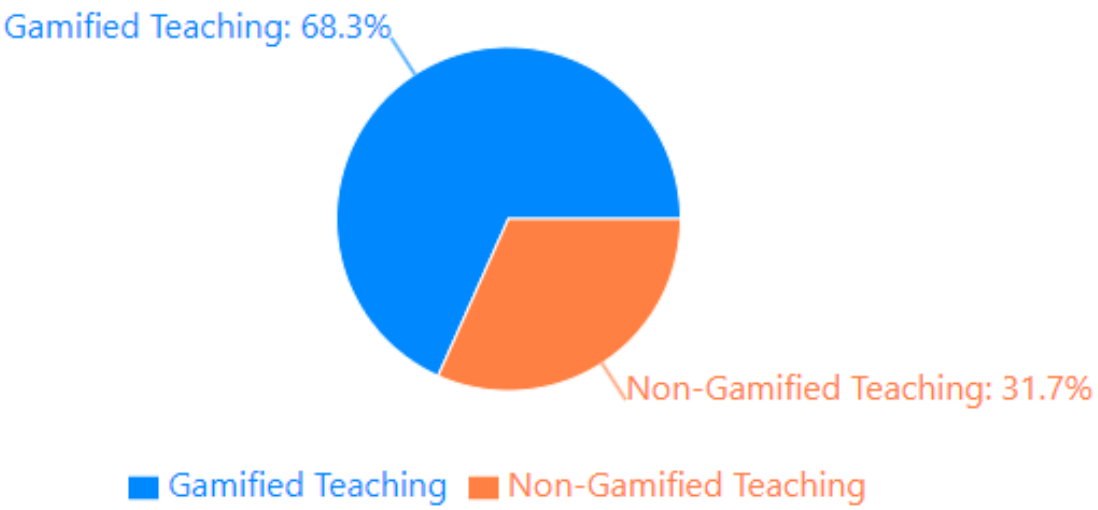


Figure 2.2: Student Engagement with gamification [1]

Table 2.1: Challenges and Constraints in Gamified Learning Systems [9]

Challenge/Constraint	**Description**
Superficial Engagement	Students may focus more on earning points and rewards rather than actual learning
Balancing Complexity and Accessibility	Gamified systems need to be inclusive and accessible to all learners with varying technological skills
Concerns of Intrinsic Motivation	Over-reliance on extrinsic rewards may diminish students' natural curiosity and desire to learn

Table 2.2: Advantages of Leveraging Gamified Learning Systems [9]

Advantage	**Description**
Increased Student Immersion	Interactive elements make learning more enjoyable and captivating, providing immediate feedback and a sense of accomplishment
Enhanced Learning Outcomes	Learners are more likely to stay motivated and achieve higher levels of mastery when learning experiences match their abilities and interests
Encouragement of Collaboration	Features like team-based challenges and peer-to-peer competitions encourage students to work together and Promotes interaction and teamwork
Analytics for Continuous Improvement	Data collected provides insights into student behaviour and learning patterns
Improved Retention Rate	More interactive and enjoyable learning experiences increase study times

AI and Gamification Combined

A key area for research entailed the unique relationship between gamified systems with AI. These two together present an opportunity to develop far more personalised and engaging learning experiences. [13] AI could be used to analyse a students performance and adapt game features like difficulty, provide immediate feedback or offer custom recommendations. [12] On the flip side, analytics that are driven by AI can provide teachers with valuable data and insight into a students learning patterns, helping them make informed decisions. Studies show that careful implementation of AI into gamified education has the chance to optimise the experience of learning, with improvements found academically. This is likely thanks to the personalised touch AI offers, enhancing engagement and motivation.

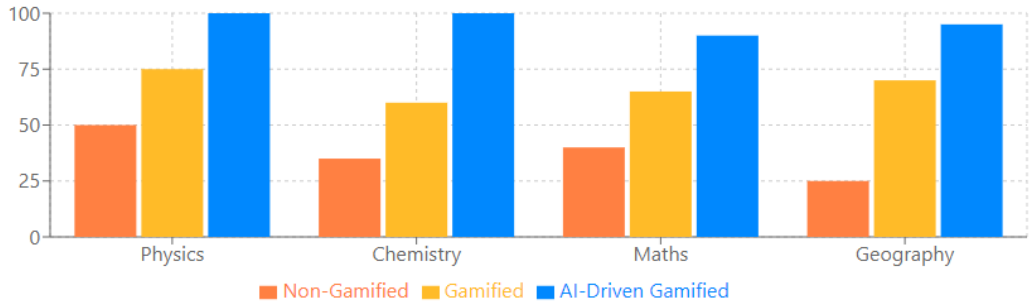


Figure 2.3: Average grades across different teaching approaches [1]

Chapter 3

Technology Review

This chapter focuses on providing a comprehensive overview of the technologies employed and used throughout the project's development. This includes the reviewing of literature for the purpose of rationalising and understanding decisions made within the systems design and implementation. Each technology is constantly aiming to link back to the project's core goal, which is to create an engaging, personalized learning platform tailored to a student's needs that leverages gamification and generative AI for an interactive study experience.

3.0.1 Microservices Architecture

Microservices are a style of architecture that involves dividing elements of an application into isolated manageable chunks. They are intended to be fairly small and independent. They make use of protocols like HTTP to communication over a network to other select elements of an application. [15] This architecture contrasts against the monolithic approach where all application functionality is maintained and developed within a single large scale application. [15] This is a single point of failure, which the Microservice approach inherently avoids falling prey to. This encouraged the decision to divide up each key functionality of the GLE into a microservice, such as Quiz Generation, User Management or Gamification and implement them as independent microservices.

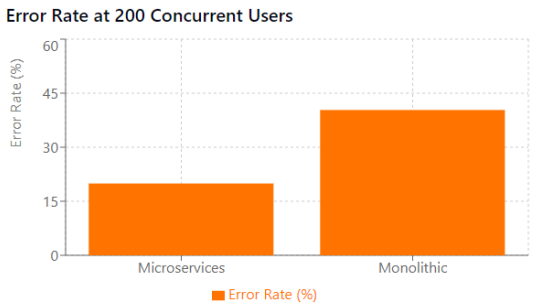


Figure 3.1: Microservices vs Monolithic Approach error rates [2]

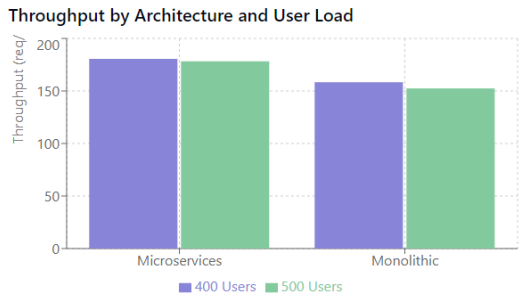


Figure 3.2: Microservices vs Monolithic Approach Throughput [2]

The benefits this architecture offers are numerous. The modularity it allows for each service for one. Each service has the ability to be developed, deployed and scaled by itself. [16]. Updates or changes to the system can then be done to one service, without affecting the greater application. This allows for more frequent work with fewer issues or disruptions throughout the process. Scalability is also a large factor that is enhanced when microservices are at play. They can be scaled up or down based on demand. [16] For example, if the quiz generation service begins to experience high load, it can be scaled up (by deploying more instances of it) without affecting other services of the platform. [15]. As stated previously, fault tolerance enhances with microservices. [15]. On a lesser note, the flexible nature this architecture provides in regards to the technology stack encourages innovation to adapt the platform in the future [15].

3.0.2 Backend - Python Flask vs FastAPI

The backends needed to be lightweight. This is where Python came in as a perfect option. However, two frameworks fit the job at hand with the choice being between Flask and FastAPI. These are both lightweight frameworks that are efficient for developing RESTful APIs and couple well with building microservices. [4] REST APIs (Representational State Transfer) are a style of development architecture for designing applications. They offer the platform a clear standard on how to communicate between the various environments of the microservice architecture without hassle. [4]In short, they rely on a stateless nature to your communication protocol and a client to server relationship. This is typically handled through HTTP using standard methods such as GET, POST, PUT and DELETE to interact with resources in a variety of ways. [4]

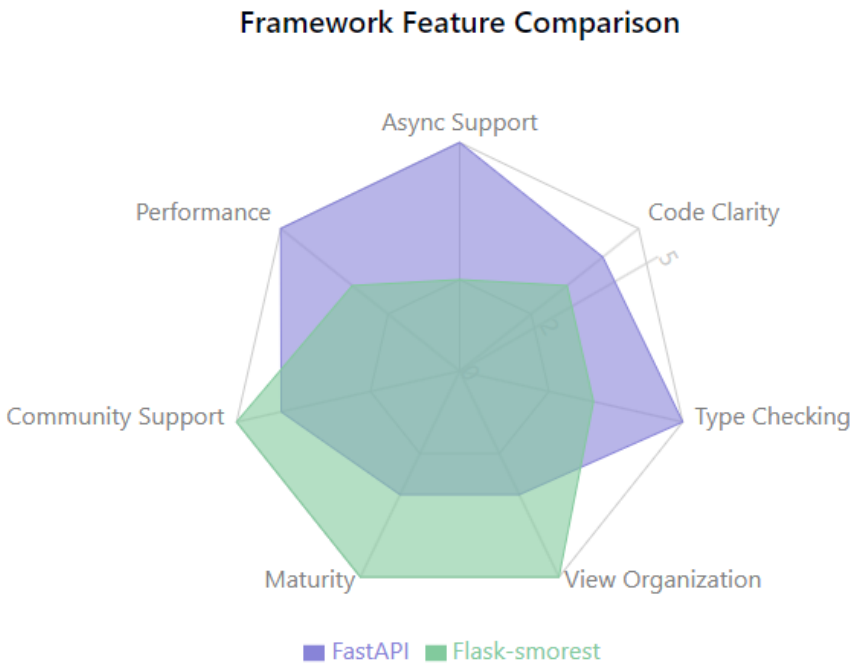


Figure 3.3: A comparison between Flask and FastAPI features [3]

A comparison between Flask and FastAPI features [3] Flask is a micro web framework for Python that is known for its simplicity and extensibility. It allows developers to construct web applications and APIs with little overhead necessary. [?] On the other hand, FastAPI lacks in the simplicity present in Flask, but offers a modern, high-performance web framework for APIs. It does offer automatic data validation, serialization as well as API documentation which could prove useful. [3] Both frameworks prove to be well-suited for the task of creating the numerous backend services of the GLE. They both provide the tools necessary for defining endpoints for the frontend for accessing quiz data, managing user information or tracking results. In the end, Flask was chosen to keep things lightweight and manageable. Flask's easily extendible nature could prove a great benefit as the project developed.[3]

The book "Microservice APIs: Using Python, Flask, FastAPI, OpenAPI and more" [3] proved a great help in exploring the implementation and comparison of these Python frameworks in regards to Microservices and Restful API communication.

Table 3.1: Framework Recommendations by Use Case [3]

Use Case	**Preferred Framework**	**Reason**
High concurrency API	FastAPI	Native async support offers better performance under concurrent load
Large codebase with complex routing	Flask	Class-based views offer better organization
Team familiar with traditional web frameworks	Flask	More familiar synchronous programming model
I/O bound microservices	FastAPI	Async processing ideal for waiting on external service

3.0.3 Frontend - Next.js VS Vue

Frontend is a vital element to the whole applications architecture. Its crucial that a platform focused on gamification, engagement and more delivers a seamless smooth experience. The top options that could deliver in this regard are Next.js and Vue.js. These are both JavaScript frameworks that offer dynamic and responsive user interfaces with no shortage of support. [6]

Next.js is a React framework that enables server-side rendering (SSR) and full stack web application creation. [4] Server side rendering improves page load times initially, which could prove highly beneficial for a Gamified platform where user experience is crucial. SSR also benefits search engine optimisation. It follows a component-based architecture, where the UI is built around different component that can be reused in other places. This promotes modularity within the frontend's codebase, keeping the whole project cohesive and maintainable. [4] It offers additional features like automatic code splitting and an optimised system for loading assets, which both contribute to faster load times and efficiency within the platform. [4]. Lastly, Next.js features a wide array of extensions and couples well with UI component providers like ShadCN/UI and Tailwind CSS for additional aesthetical options.

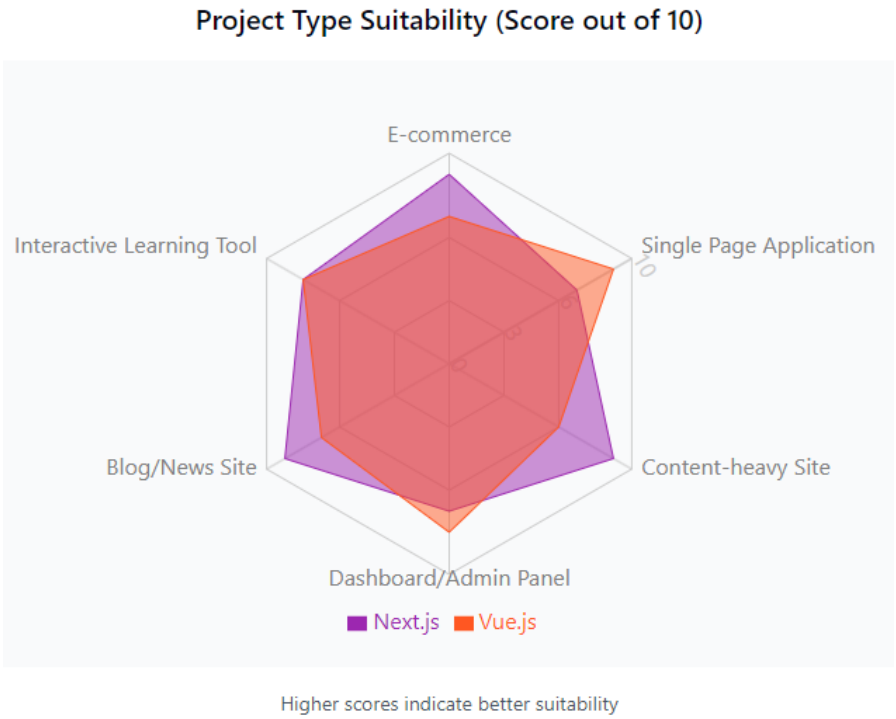


Figure 3.4: A comparison between Next.js and Vue.js based on project type [4] [5] [6]

On the other hand, Vue.js is a progressive JavaScript framework for user interface building. Similar to Next.js, it follows an architecture focused on components, but it's also known for its reactive data binding. This can be used to simplify interface updating when data changes. [5] This could make the display of results from quizzes or tracked progress within the application smoother and more dynamic. However, Vue.js is primarily focused on frontend development. While it can be integrated with backends through API calls, it leaves options far more limited. [5]

Ultimately its a choice between Next.js's more flexible capabilities, with the option for full-stack should it prove necessary and the enticing server side rendering or a potentially smoother handling of data changes with Vue.js. Considering the project at hand and the possibility of changing needs as work progresses, Next.js proves to be the logical choice. Additionally, its greater support overall, its adjacency to base React functionality and smooth coupling with offerings like Tailwind and ShadCN/UI are an additional benefit that will make the overall development of the gamified learning environment smoother with Next.js as the primary development platform for the frontend. [6]

Metric	Next.js	Vue.js	Advantage
Average Performance Score	99.4	75	Next.js by 32.5%
Average Speed Index	0.8s	1.1s	Next.js by 27.3%
Average Time to Interact	1.44s	1.12s	Vue.js by 28.6%
Total Lines of Code	269	231	Vue.js by 16.5%

Figure 3.5: Performance metrics: Next.js vs Vue.js [6]

3.0.4 MongoDB - NoSQL Document Based Storage

The application utilises MongoDB for its choice of database storage. MongoDB is a NoSQL (Stands for Not Only SQL) document database. It's designed with scalability and flexibility in mind, making it a convenient choice for this project. [17] It is unlike traditional methods of database storage such as relational databases where data is stored in a series of generated tables with set schemas. Mongo offers a flexible JSON style of document storage that utilises dynamic schemas instead. [17] Document-based storage, particularly JSON-like, is well-suited for the task at hand on this platform. It allows for convenient storage of various types of data within the GLE whether that be personal user information, gamified player stats, generated quiz data or tracked results. Overall, it allows a flexible freedom during the development for handling the different evolving data requirements present.

This namely comes in it's flexibility, where it allows for easy changes of data structures when required. [17] It's highly scalable in nature, capable of handling larger volumes of data or a higher traffic load than other options on the market. [17] This aligns well with the likely higher loads of data a gamified learning platform like this might require. Its document storage aligns quite naturally with OOP (Object-Oriented Programming) paradigms, which could prove beneficial in simplifying elements of development. Lastly, Mongo's built-in support for key-value pairing is perfectly suited for fast data access, which is crucial for the gamification microservice where passing and grabbing of achievements, badges and leaderboards should be quick and well managed. [17]

3.0.5 Generative AI Models - Quiz Generation

One of the core features of this Gamified Learning Environment is the implementation of generative AI models such as Open AI's GPT (Generative Pre-trained Transformer) or Anthropic's Claude AI for dynamic quiz generation. Generative AI as a whole, but more particularly large language models (LLMs) have proved through demonstrations a unignorable ability to understand human text and generate their own liking of it. [18] [19] In the context of this Gamified Learning Platform, these models are being utilised for customised dynamic quiz creation based on options selected by the user (fed through prompt engineering), their own notes and/or uploaded pdfs that are parsed, extrapolated and fed into the platform through the quiz generation microservice.

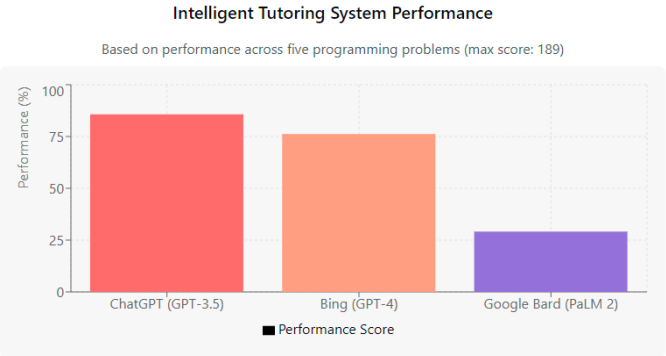


Figure 3.6: Intelligence comparison between Generative AIs [7]

This use of generative AI for the production of quizzes offers several noticeable advantages. Firstly is sheer customisability. It allows the creation of a wide variety of quizzes at a moments notice from a user's own uploaded and/or selected material. This hopefully aids students in the process of studying, providing a better method of study for them where their own notes influence the creation of quizzes catered to their needs. The AI's can have their returned quiz adapted and tailored to a student's needs or skill level. This could be accomplished through selectable parameters like difficulty levels of "Beginner", "Intermediate" and "Expert" or the scaling of the number of questions generated. Structured Prompt Engineering is highly implemented to ensure an accurate generation based on the user's requirements. Generative AI also assists through the sheer speed in which it can create an assessment quickly and efficiently. This can be beneficial for both learners for studying or testing themselves on their knowledge base and potentially for educators where a teacher could create a quiz from their own study material and invite students to attempt it. [18]

It is important however to be aware and acknowledge that AI-generated content can contain inaccuracies or lack complete originality, the GLE tries its best to use it as an augmentative tool for enhancing quiz creation, but at best it should be used alongside real study to verify your learnings. In an aim to reduce and/or alert the user to inaccuracies or sub-par generated questions that may occur, the platform implements the use of AI for cross-validation prior to return to the user. Finally, it's worth mentioning that in an effort to avoid reliance on one singular generative AI, several are implemented in the quiz generation process, and are selectable by the user prior to question generation, allowing for comparisons in generation results.

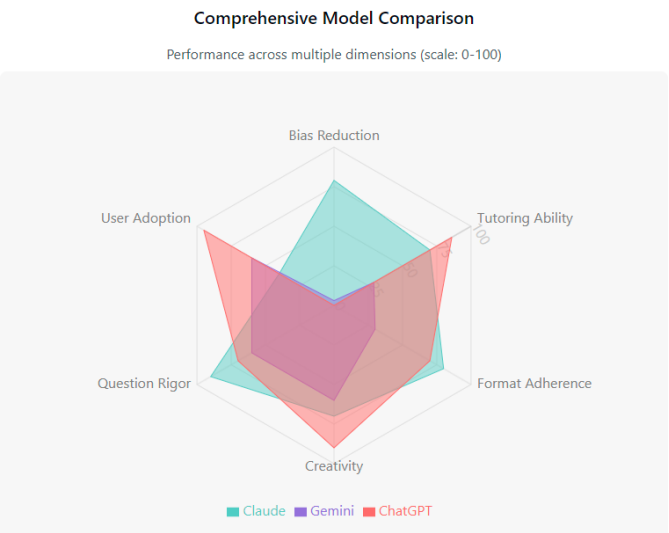


Figure 3.7: Capabilities comparison between Generative AIs from own findings

3.0.6 Prompt Engineering Techniques - Personalised Quizzes

Prompt engineering is the process where one designs effective prompts or instructions for a generative AI to adhere to, all in the aim to return a desired output. [20] In the GLE Platform, prompt engineering is crucially utilized for generating personalized quizzes. Inputs to an AI model are delicately crafted and locally tested so as to provide the desired output. Through the use of this, the system can guide the model into creating quiz questions that are relevant to a specific topic, aligned to a particular difficulty level or number of questions, and potentially tailored to a student's past performance.

An Example:

A user may specify they want to make a chemistry quiz based on the material from the current topic being covered in class. They specify title, description, etc and select AI generation. They choose which model and set constraints such as how many questions to generate or the difficulty level. They then can

either manually paste in any notes or upload the PDF of their class material and hit "generate". The system will then use this data and fill in the blanks on its constructed back end prompt, allowing the AI model to generate the appropriate quiz questions. A greater aim would be to utilise stored and accumulated data to further personalise the quiz generation process based on their past results in the quiz category or noticed learning patterns. This would all be in the aim of creating a learning environment that is adaptable to a student's individual needs through the use of these carefully structured prompts.

3.0.7 Results Graphing

The Gamified Learning Environment would benefit greatly from the inclusion of results graphing and data visualisation of one's study efforts. D3.js (Data Driven Documents) is a key technology for this. It is essentially a JavaScript library that is designed for creating and manipulating documents based on data. [21] This allows for a wide variety of graphics to be developed for displaying information within a web browser. [22] [23] How it operates is this: it loads data into a browser's memory and binds it to the DOM (Document Object Model). That data is then transformed using HTML, SVG and CSS to generate the graphics. [21] It allows data to be presented on a site vividly and intuitively [23] and it's flexible and extendible for creating all kinds of types of charts. [22] Additionally, it's highly compatible with most modern web browsers and adheres to web standards.[23]

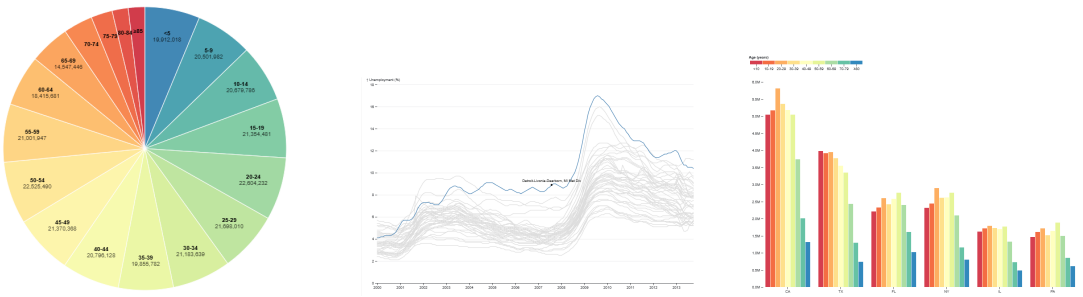


Figure 3.8: Three examples of Graphics made with D3.js [8]

Within the GLE, D3.js could be integrated within Frontend components and within the Results Tracking microservice at specific points. This allows for the generation of charts and graphs for user progress, quiz results or other game or performance statistics. These charts can do wonders for aiding users in tracking their learning journey and help them identify weak points.

3.0.8 Tailwind CSS and Shadcn/ui

Tailwind CSS is a utility CSS framework that offers various low-level utility classes for composing and building custom aesthetical designs in HTML. [20] Unlike traditional CSS frameworks that provide select pre-built components (such as Shadcn/ui, also used throughout this project), Tailwind CSS uses a more customisable approach to web application styling. [20] It is implemented throughout the applications frontend to enhance the aesthetic and visuals of the platform within the gamified features and ensuring a responsive user interface.

Responsive user interfaces are important to maintain and develop for modern web applications such as this one. It should adapt seamlessly to different screen sizes and devices such as desktop or smartphone. Their lightweight nature allows for convenient coupling with Next.js and their utility classes aid in building UI by providing tools and syntax for controlling element spacing, typography and more granular responses. This is utilised heavily for the creation of clean appealing interfaces for the gamified elements of the GLE such as the achievements, leaderboards, progress trackers, badges and more.

3.0.9 Docker and Kubernetes - Containerisation and Orchestration

Docker and Kubernetes are industry leading technologies known for containerisation and orchestration of microservices. [24] Each microservice in the GLE platform will be containerised using Docker, ensuring there is a consistent deployment across different environments and in an effort to simplify the set up process. Containerisation enhances the portability as containers can run on any system with Docker installed. Furthermore, it improves the degree of isolation between micro services, ensuring they are independent in their infrastructure and how they're handled, as well as separating the Frontend. [25]

Kubernetes is a technology utilised for the orchestration of containers, its a platform that helps automate the deployment process as well as scaling containers and the general management of containerized applications. [25] In the GLE, Kubernetes was aimed to be utilised for managing the various Docker containers at play. It would allow for scaling them based on demand or need, and allow for handling tasks such as load balancing and update rollouts. Kubernetes infrastructure could prove increasingly valuable as the project develops in size for maintaining the availability and scalability that the platform may need if deployed in a cloud-native environment. [25]

3.0.10 REST APIs for Communication

REST APIs serve as the primary way of communicating between the frontend and the backend microservices within the GLE's infrastructure, as well as for minor communications between microservices themselves. As discussed previously in Section 3.2, REST is an architectural style that involves HTTP methods interacting with resources. In the GLE, the Next.js frontend will make numerous API calls to each of the Flask backend microservices throughout its operations. This will be done for data requests such as requesting quiz data for questions, user profile information or game stats. This will also be done for pushing or sending data such as quiz answers, player results, profile updates or more. [4]

RESTful communication ensures that the platforms architecture stays loosely coupled, as wanted. This would be where the frontend and backend can evolve and adapt independently as long as they maintain their relationship. [4] The use of JSON (JavaScript Object Notation) as the standard data format for this communication across APIs allows for easy transmission of data between different sections of the application. [26]

3.0.11 Gamification Principles and Features

The crossover between gamification and education is an involved approach to developing a software as a service. It involves the application of game design elements and gaming principles in teaching and learning environments in an effort to increase a student's motivation and engagement to learn and study. [27] [11] The GLE platform has gamification heavily incorporated not only into its inspiration, but also its principles. Gamified Principles are applied throughout the platform to create a more enjoyable, effective learning experience that complements traditional studying in an effort to keep student's studying longer.

The list is constantly evolving, but several gamification features being implemented into the GLE include:

Table 3.2: Potential Gamification Features

Feature	**Description**
Achievements	Recognising when a student accomplishes substantial study goals and rewarding them for their efforts. This could be for completing tasks, reaching milestones or completing a number of quizzes in a category or achieving high scores. Rewards involve experience points for a consistent levelling experience and badges for flaunting a student's accomplishments.
Leaderboards	Displaying a ranked list of users based on their performances or engagement across various stats. This aims to foster friendly competition and encourage others to study harder to rise up the leaderboard. [5]
Challenges	Rotating challenges for a user to partake in. Completion of these challenges would ideally grant a reward such as experience points, incentivising the user to keep learning.
Campaigns	Set study paths for a user to select and progress through. These could help a user find a path for what to study when they're suffering decision paralysis.
Levelling System	The integration of levels across the system could do wonders for encouraging a user to grow and develop as a learner on the platform.
Categories	Quizzes can be broken up into select categories based on the topic they cover or realm of study they entail such as Chemistry, Mathematics or Engineering. Completion of quizzes within these categories offers category specific experience points. This encourages users to engage with their chosen topic and earn category levels.

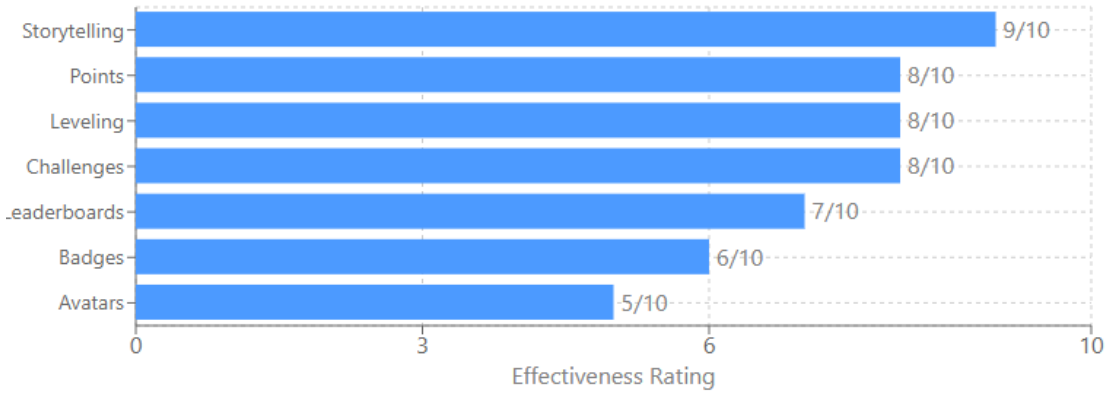


Figure 3.9: Effectiveness rating of different gamification techniques [1]

These features are all designed to reward study effort and enhance someone's motivation to learn. [27] The gamification service will manage experience points, badges, achievements and the gamified side of a user's stats such as experience points earned, category levels achieved, achievements earned, etc. The goal here is to create a platform that is far more engaging than typical forms of study offered to a student.

3.0.12 Standards and Common Protocols

Across the development of the GLE application, adherence to relevant standards should be considered at all times. For RESTful API development, following the standards for HTTP and best practices for API design is necessary for interoperability. [4] In regards to Data exchange, JSON is widely adopted as an industry standard. [4] [26] In terms of general software development practices, following established coding conventions and utilising version control systems through GitHub is essential for maintainability and collaborative development. Containerisation through Docker relies on container image formatting standards, while Kubernetes should follow its own set of standards for managing clusters and orchestrating containers.

3.0.13 Conclusion

This chapter has provided this paper with a detailed overview into the key technologies involved in this GLE platform, including discussions over comparable options and decisions made. The choice of a microservices architecture was explored, backend frameworks like Python Flask vs FastAPI, frontend frameworks

such as Next.js or Vue.js, and the NoSQL database MongoDB vs traditional SQL. Generative AI models from OpenAI and Anthropic as well as prompt engineering techniques. Styling with Tailwind CSS and shadcn/ui was lightly touched on as well. Containerisation with Docker and orchestration with Kubernetes were discussed in the context of the project's aim to develop a modular, scalable learning environment. An in-depth analysis of gamification principles and specific features was undergone to better integrate them for engaging student's and enhancing their motivation to learn and interact with the platform. From this point, the subsequent chapters to come will use this as a foundation to build upon for describing and detailing the design and implementation of the GLE platform in detail.

Chapter 4

System Design

This chapter offers a full comprehensive analysis of the overall system architecture for Exper, the Gamified Learning Environment (GLE). This chapter will detail the technical implementation, design decisions, and where or how different components interact with each other to develop a learning platform that is cohesive and incorporates user management, gamification, quiz generation and results tracking.

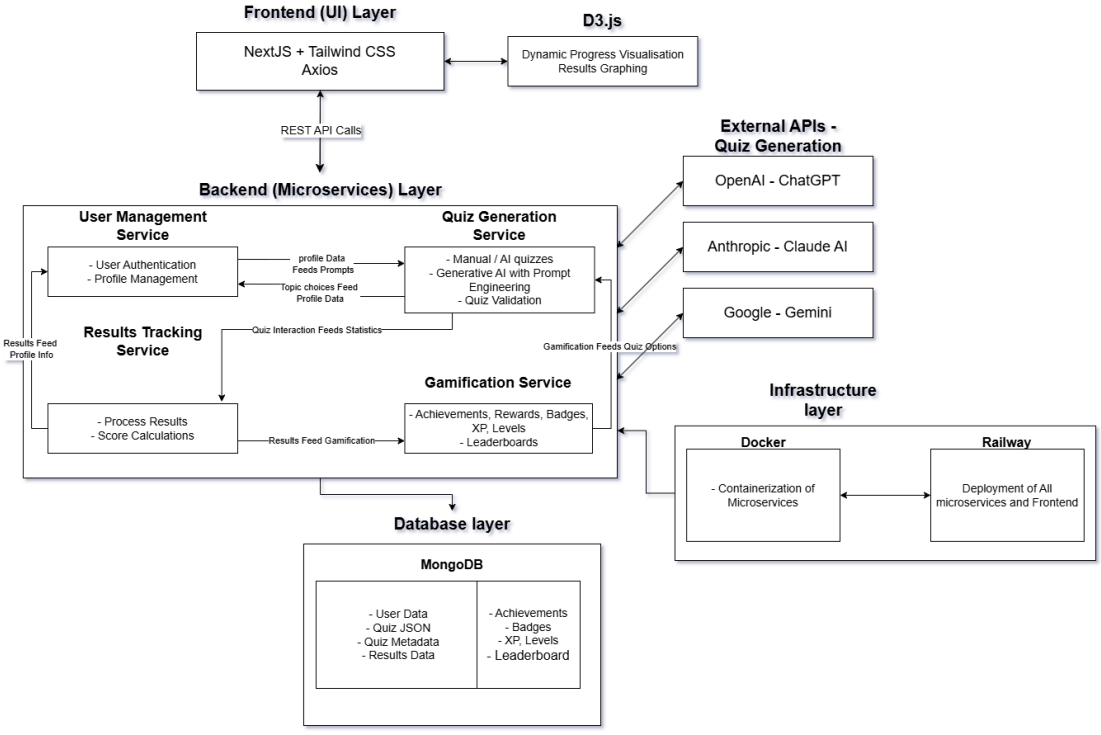


Figure 4.1: System Architecture.

4.0.1 Overall Architecture

Exper employs a modern, distributed architecture focused on a Frontend with a series of microservices. This ensures modularity in the codebase, optional scalability and makes them easier to maintain. Each microservice handles distinct responsibilities within the platform, filling one of its many needs. The general architecture follows a client-server model where the Next.js frontend serves as the interface that is client facing, while the each backend microservice handles specialised functions. Communication across these components occurs exclusively through RESTful APIs. This ensures an adherence to loose coupling and keeps scalability independent of each component.

As shown in Figure 4.1, the architecture uses containerisation through Docker, with each microservice (and the frontend) encapsulated in their own containers. The aim was to further orchestrate these using Kubernetes to enable automated deployment, scaling and greater management of the application. Attempts were made to integrate Kubernetes into the projects infrastructure, but ultimately fell out of scope and was deemed unfeasible.

Whether the system only uses Docker or managed to include Kubernetes, this ensures that the GLE is scalable and resilient to partial crashes or varying load efficiencies. In fact, it was found that the Quiz Generation microservice required a sizable increase in service workers needed than the other services when deployed for handling the Generative AI API communications. The use of Docker helped scale this service accordingly, allowing it to increase its worker load without affecting the rest.

4.0.2 Component Overview

Next.js Frontend

The frontend for Exper is built primarily with Next.js. This is a modern React framework that enables server-side rendering, client-side rendering, and static site generation as needed (which will prove very useful). The hybrid approach this offers grants highly noticeable benefits to performance and enhance the user's experience. Furthermore, Next.js integrates well with various UI components that allow for the development of the system's gamification.

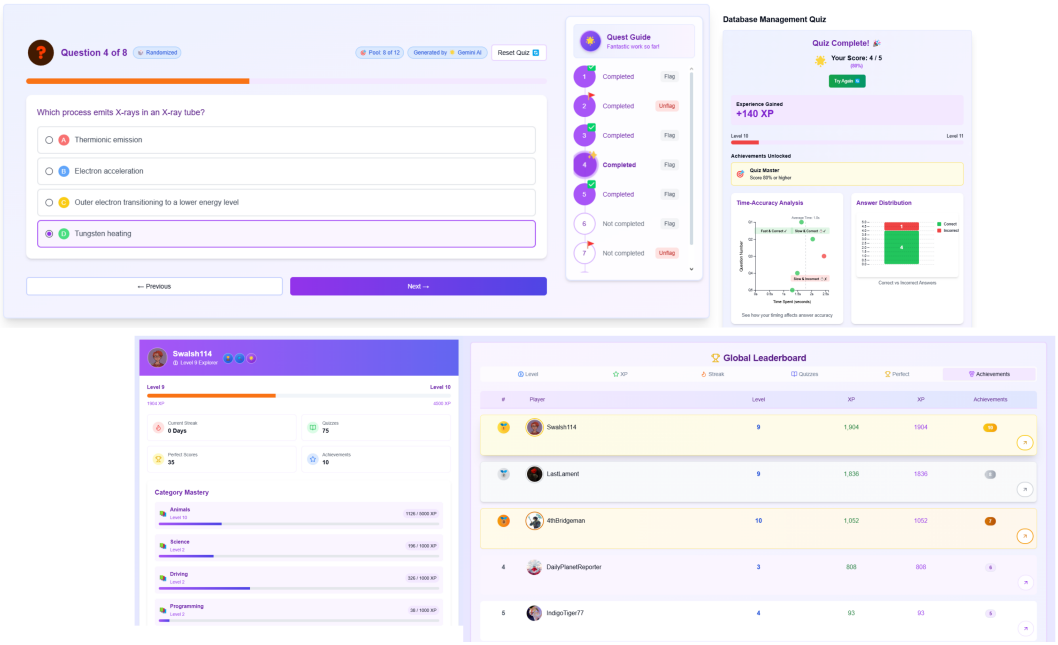


Figure 4.2: Showcase of various aspects of the Frontend

To assume the Frontend developed is simple would be a vast understatement, it is quite involved in its functionality. This includes but is not limited to:

1. Component Structure - The UI uses a hierarchical structure for its components. This involves reusable components for various common elements such as quiz cards, leaderboards or achievement displays. This promotes consistency across the platform and simplifies maintenance. Many of the components are separated out by the hooks for functionality they use, by the types they make use of and by the html output they return to the user.

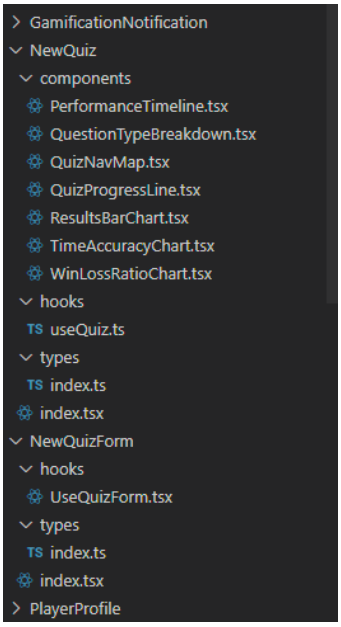


Figure 4.3: File hierarchy for components

2. State Management - The state of the application is managed using both React's Context API for global state and local component state for more UI-specific interactions that take place. This is done in an effort to create a balance between performance and ease of development.

3. Styling - Tailwind CSS is utilised for utility-first styling. This is combined with shadcn/ui to allow for consistent UI components that were accessible and reusable. Additionally, this ensures that the platform is responsive across different devices. Styling was an element of this platform that was constantly adjusted as it developed in an effort to present the best layout to the user.

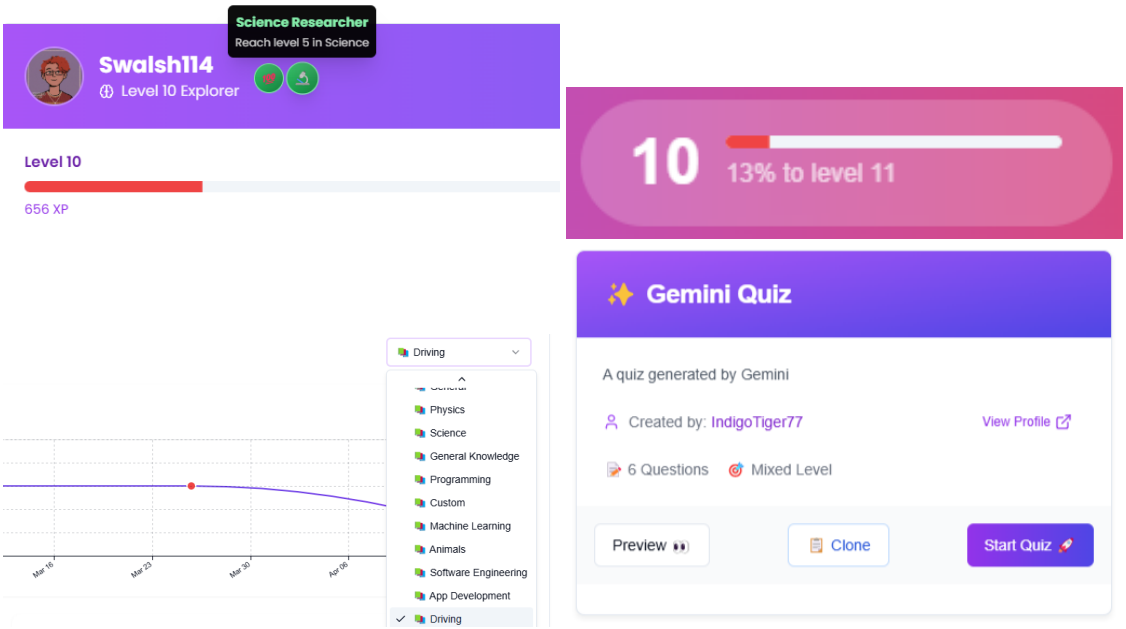


Figure 4.4: Showcase of various aspects of the Frontend

4. Data Visualization - D3.js is integrated in select points for the generation of charts and graphs for user progress, quiz results and other larger performance metrics. Visual graphs like this aid users in tracking their learning journey and do wonders for identifying their weak points or room for improvements. They are used primarily in two places on the frontend: Quiz results display and category progress. More on this later in the Results Tracking Microservice section.

5. API integration - Custom hooks and minor services are created for encapsulating API call handling to the backend microservices. These help with handling authentication, data fetching or caching and safely handling errors. This works to simplify and apply a level of abstraction to data access throughout the application.

```
const response = await fetch(endpoint, { // Fetch quiz from API
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ // Send question count, difficulty, and notes
    notes: notes,        You, 2 months ago • Completed QuizForm modularity
    pdfUrl: pdfUrl,
    parameters: {
      questionCount: questionCount, // Number of questions to generate
      difficulty: difficulty, // Difficulty level of questions
      includeExplanations: true, // Include explanations for answers
    }
  })
});
```

Figure 4.5: A typical API call on the frontend to a microservice

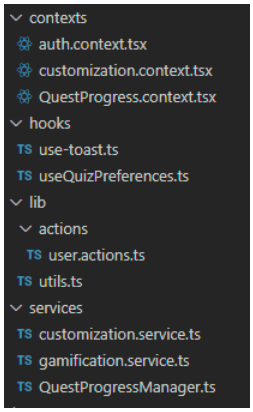


Figure 4.6: File hierarchy for contexts, hooks, services and more that interact with backend

Quiz Attempts

A core component I'd like to highlight from the Frontend is the active Quiz component. This is where a user attempts a quiz, navigating through each question until they receive their results at the end. Its complete with a progress bar, question selection, details on the current quiz, reset functionality as well as a quest guide for navigating between questions and marking them completed or flagged for coming back to.

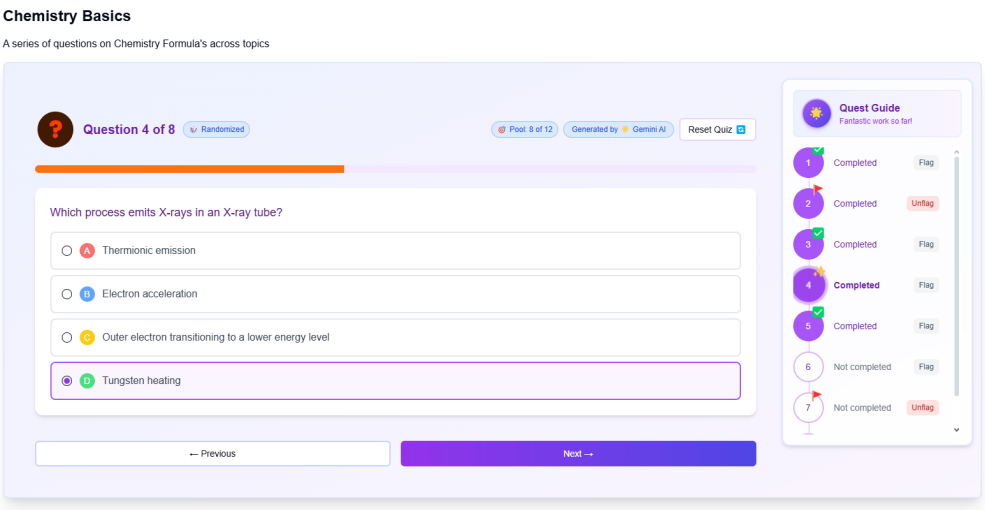


Figure 4.7: A Quiz Attempt in Exper

Note

For the sake of brevity this is in no way a showcase of the entire codebase for the Frontend of Exper, but only a horizontal slice through its many aspects. Many components of the Frontend will be touched on again within each microservice in regards to their interaction. For a full review of the codebase it is encouraged to check out the Github.

User Management Microservice

The User Management microservice handles user creation, authentication, authorisation and general management of users within the GLE. Its built using Python with Flask. During its development, user security was kept in mind. From the beginning it was obvious that correctly handling and storing user's data was crucial for a gamified quiz platform like this, so it became a core pillar of the architecture.

Key features include:

1. Authentication System: JSON Web Tokens are implemented for secure, stateless authentication. The system supports email and password authentication and full management of the user's state.

2. User Profiles: Manages user data including personal information, profile images and names. Profiles are stored in MongoDB, making use of its flexible document structure in case requirements shifted during development.

3. Authorisation: A degree of authorisation is implemented through this service, whereupon the frontend must send a RESTful request to authorise the existence of a user within the database before allowing them to access the services.

4. Security Features: For user security and peace of mind, incorporates password hashing and password hash checking with Werkzeug for securely storing passwords without putting the user's account at risk.

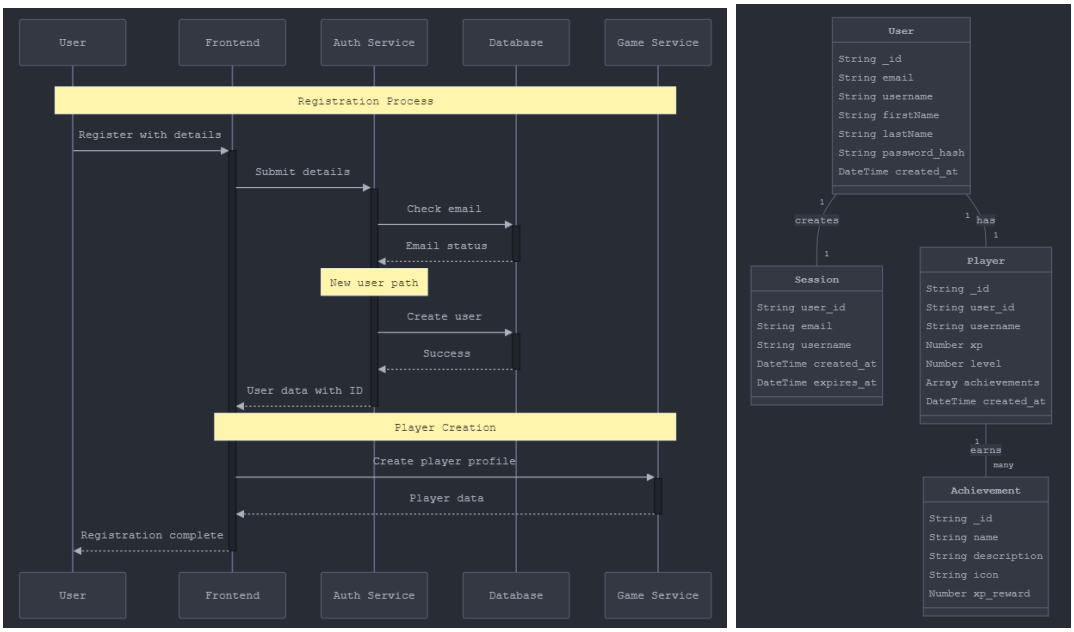


Figure 4.8: Sequence Diagram: User creation process

Figure 4.9: User data model

```python
# /register route to create new user in database
@auth_bp.route('/register', methods=['POST'])
def register():
    try:
        data = request.get_json()

        # Validate required fields
        required = ['email', 'password', 'username', 'firstName', 'lastName']
        if not all(field in data for field in required):
            return jsonify({'message': 'Missing required fields'}), 400

        # Check if user already exists
        if db.userdb.usercollection.find_one({'email': data['email']}):
            return jsonify({'message': 'User already exists'}), 409

        # Create new user
        newUser = {
            'email': data['email'],
            'password': generate_password_hash(data['password']),
            'username': data['username'],
            'firstName': data['firstName'],
            'lastName': data['lastName'],
            'imageUrl': data.get('imageUrl', ''),
            'created_at': datetime.utcnow()
        }

        result = db.userdb.usercollection.insert_one(newUser)
        newUser['_id'] = str(result.inserted_id) # Convert ObjectId to string

        return jsonify({'message': 'User created successfully', 'user': newUser}), 201
    except Exception as e:
        print(f"Error: {e}")
        return jsonify({'message': 'Internal Server Error'}), 500
```

```python
# /verify route to authenticate if user is logged in
@auth_bp.route('/verify', methods=['GET'])
def verify_session():
    try:
        if 'user' not in session:
            return jsonify({'message': 'Unauthorized'}), 401

        user_id = session['user'].get('_id')
        if not user_id:
            print("Invalid user data in session")  # Debug logging
            return jsonify({'message': 'Invalid session'}), 401

        user_data = session.get('user')
        if not user_data or '_id' not in user_data:
            print("Invalid user data in session")  # Debug logging
            return jsonify({'message': 'Invalid session'}), 401

        # Return success with user data
        return jsonify({
            'message': 'Session valid',
            'user': user_data
        }), 200

    except Exception as e:
        print(f"Error verifying session: {str(e)}")
        return jsonify({'message': 'Server error'}), 500
```

Figure 4.10: User registration code snippet

Figure 4.11: User authentication code snippet

Early on in the project's lifecycle, external user authentication and management was explored through the use of Clerk. Clerk proved extremely competent in the services it provided, including OAuth integration with providers like Google or Microsoft. However it proved far more complicated to integrate seamlessly with the platform, particularly with the Flask backend. Clerk integrates ideally with full stack Next.js architectures, which wouldn't adhere to this platform's aims in terms of modularity and independence. This encourages the move to handling user authentication within its own microservice, rather than seeking outside solutions like Clerk.

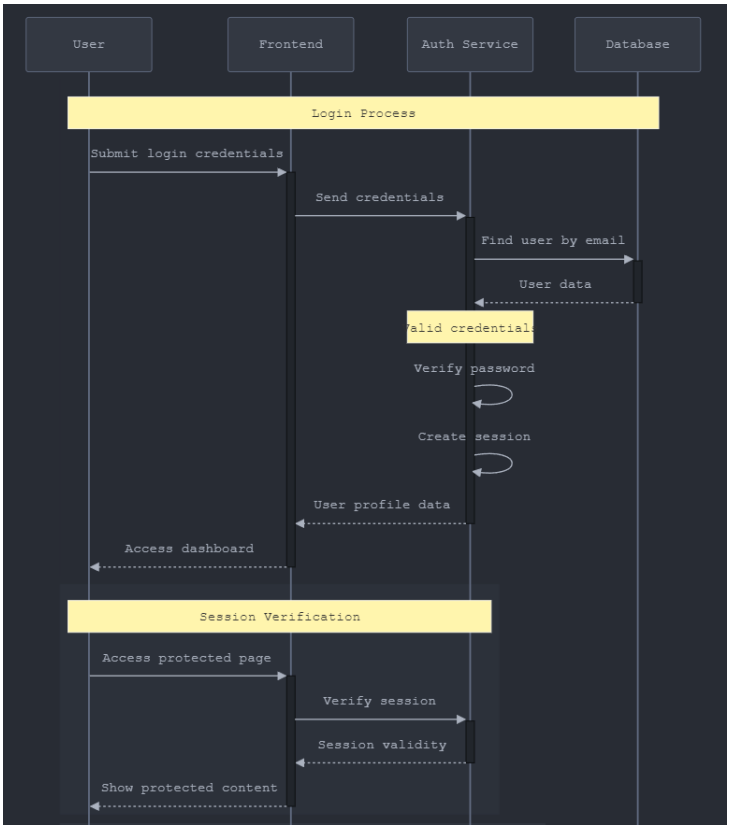


Figure 4.12: Sequence diagram: Login and Authentication process

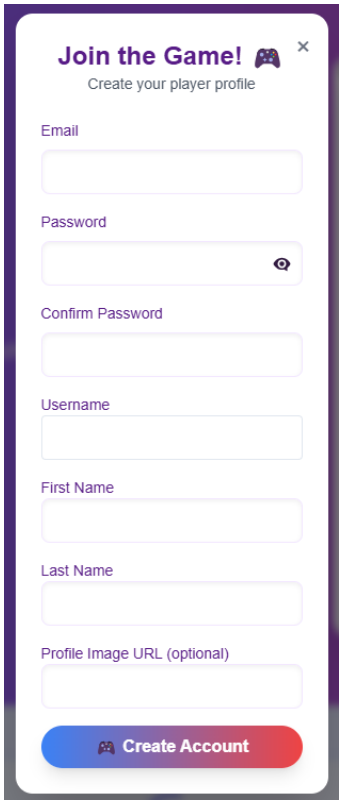


Figure 4.13: User Registration Modal

All elements of the service are exposed through RESTful endpoints for user registration, authentication and user data management. These are done using appropriate middleware to prevent unwanted access from outside parties.

Quiz Generation Microservice

The Quiz Generation microservice is the core for creating quizzes, managing them and delivering them to users. It supports both manual quiz question creation and AI powered generation through the use of APIs. This hopefully makes it flexible in its implementation. It possesses the ability to generate quizzes from either Open AI's Chatgpt, Anthropic's Claude or Google's Gemini, so its never reliant on just one and gives the users options for question generation.

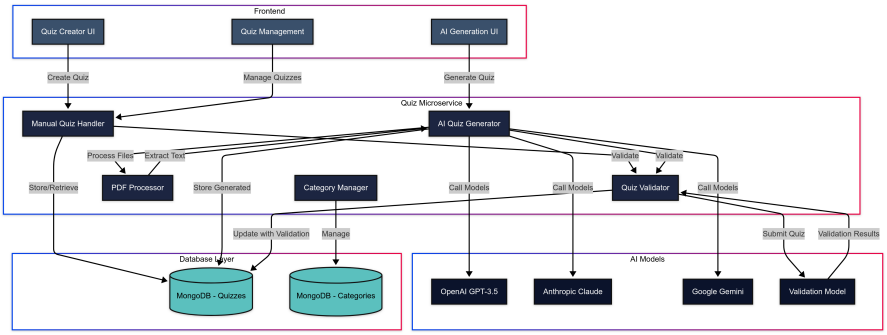
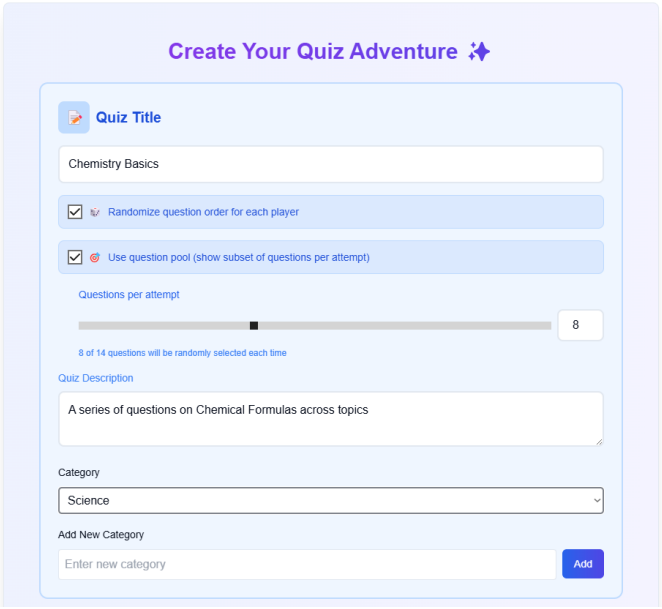


Figure 4.14: Architectural Diagram of Quiz System

- Quiz Data Model: It has a flexible schema that is designed with support for either multi-choice answers or 1 in 4 answers, difficulty levels and categories of study. All quizzes are stored in MongoDB with IDs for efficient retrieval. Quizzes possess both a QuizID and a UserId of the user that created them.
- Manual Quiz Creation: Questions, answers and metadeta can be manually inputted by the user and sent RESTfully to the backend for quiz object creation. This includes image support.
- Randomised Question Pools: In the later days of the project, question randomisation was implemented into the question creation process. This allows a quiz creator to have the quiz randomise questions upon an attempt, rather than be in a set sequential order. Additionally, question pools were implemented to complement this. This allows a quiz creator to create a larger pool of questions and have a select amount be pulled from on an attempt, similar to seen on platforms like Moodle.
- AI-Powered Generation with Prompt Engineering: Integrates with either OpenAI's ChatGPT, Anthropic's Claude or Google's Gemini for generative AI quiz creation. This is done through several methods that utilize prompt engineering. The prompt varies in approach based on the required AI's API structure with extensive testing on results giving.



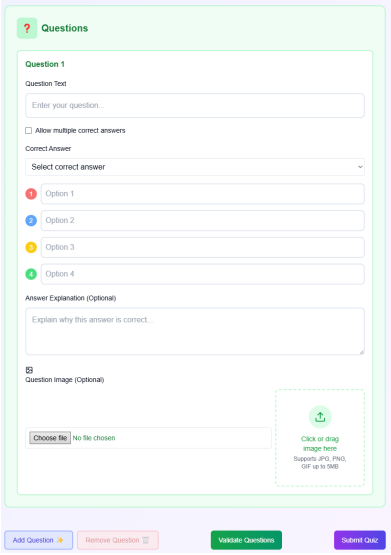


Figure 4.15: Quiz creation UI

Figure 4.16: Manual question creation UI

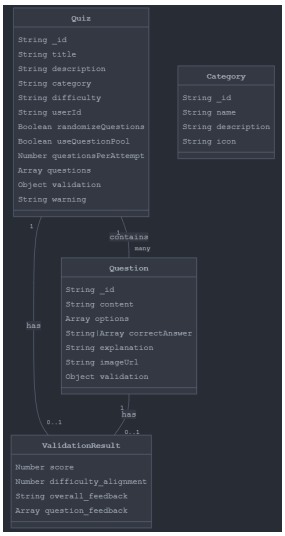


Figure 4.17: Quiz creation code snippet

Figure 4.18: Quiz Generation code snippet including prompt engineering

Figure 4.19: Quiz data model

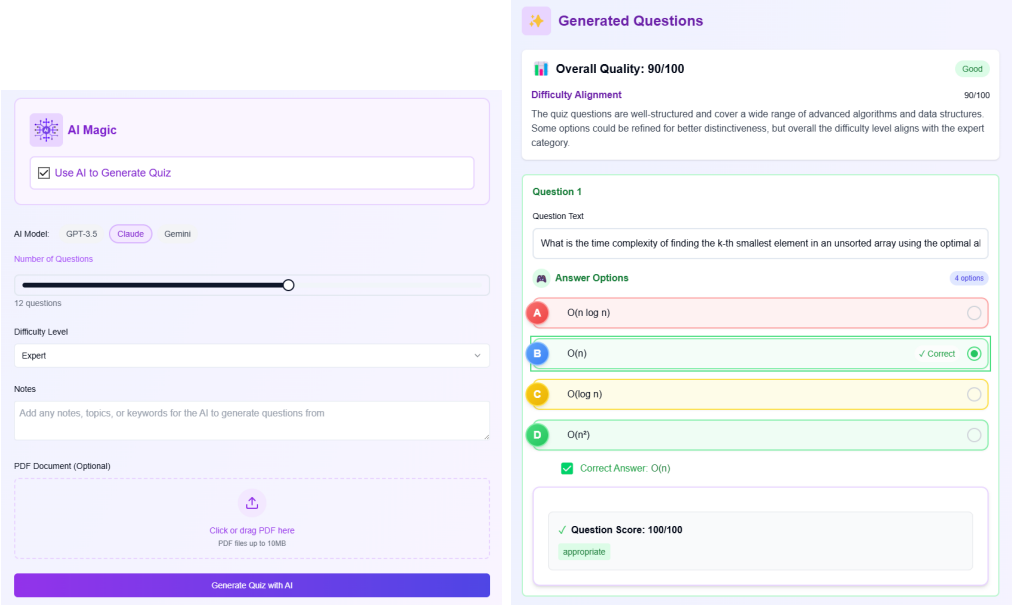


Figure 4.20: AI Question generation UI

Figure 4.21: Questions Generated for a quiz using an AI

With this functionality, the system can:

- Generate quizzes from user-provided notes or materials.
- Adjust difficulty levels based on several tiers.
- Adjust number of questions based on a range.
- Generate explanations for why an answer is correct.

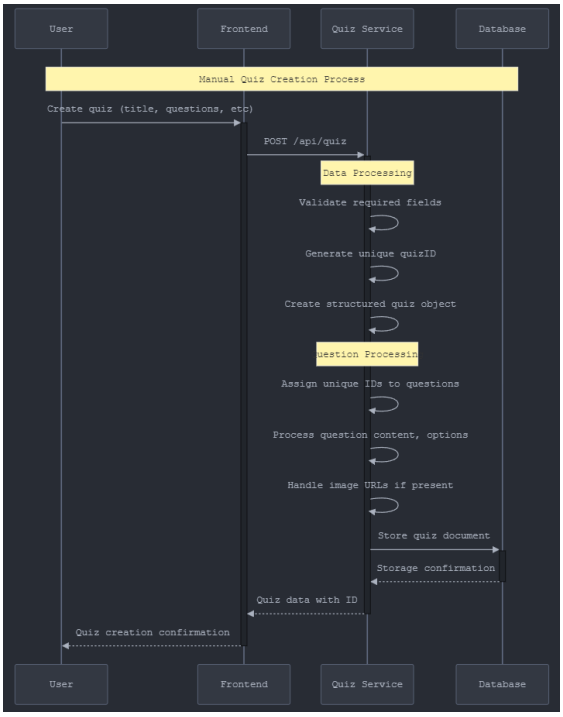


Figure 4.22: Sequence diagram: manual quiz creation process

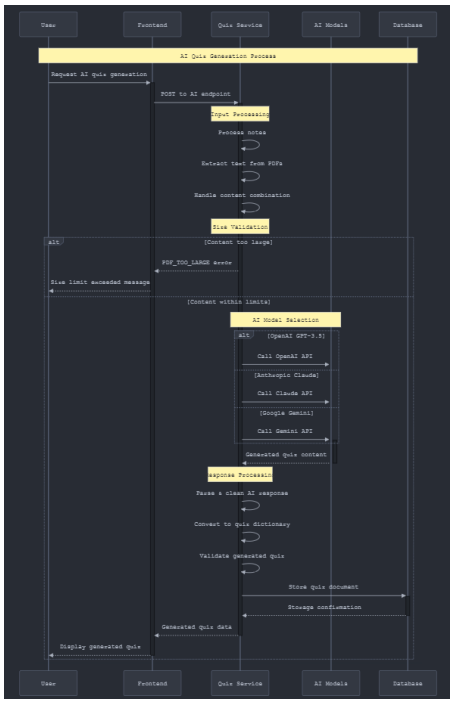


Figure 4.23: Sequence diagram: AI quiz generation process

The prompt engineering pipeline is sophisticated enough that it can transform a user's requirements into optimised prompts for any of the AI models. This largely involves templates with dynamic parameters that are inserted based on the user's passed in choices.

- PDF Parsing: For added convenience and feasibility of use. PDF's can be uploaded On the frontend and passed to the Quiz Management Microservice where they go through a digestion process for question generation. They are temporarily stored using GridFS, parsed of text and fed into the prompt instead of / alongside the pasted notes of a user.

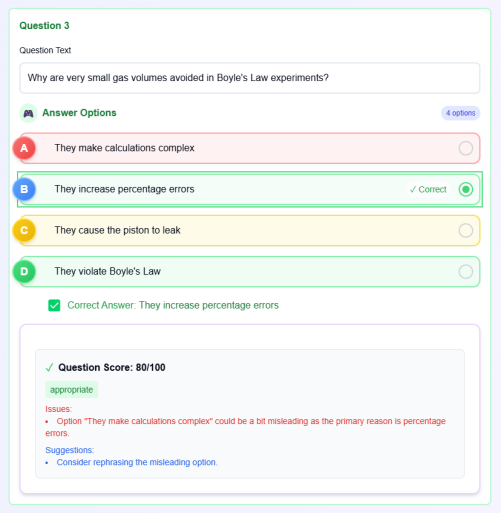
- Quality Assurance: During the Christmas seminar, it was suggested that quiz quality could be possibly ensured with the aid of AI for quality validation. This mechanic was trialed and implemented shortly after the seminar. Upon generation, questions are automatically validated based on several criteria such as question clarity, distinctiveness, educational value and alignment with difficulty.

```python
# Extract text from PDF using PyPDF2 and handle both URLs and local file paths
def extract_text_from_pdf(pdf_path, check_size=True):
    try:
        # Handle both URLs and local file paths
        if pdf_path.startswith(('http://', 'https://')):
            # For URLs
            response = requests.get(pdf_path)
            response.raise_for_status()
            pdf_file = io.BytesIO(response.content)
        else:
            # For local files - remove file:// prefix if present
            if pdf_path.startswith('file:///'):
                pdf_path = pdf_path[8:]  # Remove 'file:///'

            # Decode URL-encoded characters in the path
            pdf_path = unquote(pdf_path)

            # Open local file directly
            pdf_file = open(pdf_path, 'rb')

        # Read PDF content
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        total_pages = len(pdf_reader.pages)
        print(f"Extracting text from {total_pages} pages of PDF")
    You, 3 days ago • pdf load testing
        # For smaller PDFs, proceed with normal extraction
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"

        # Check if PDF is large
        if check_size and len(text) > 60000:
            if not isinstance(pdf_file, io.BytesIO):
                pdf_file.close()
            # Return a signal that this PDF is too large
            return "PDF_TOO_LARGE"

        # Close the file if it's a local file
        if not isinstance(pdf_file, io.BytesIO):
            pdf_file.close()

        # Check if total text is too large (approx 15,000 tokens ~= 60,000 chars)
        if check_size and len(text) > 60000:
            return "PDF_TOO_LARGE"

        print(f"EXTRACTED TEST START: {text[:100]}...") # Print first 200 characters
        print(f"EXTRACTED TEXT END: {text[-100:]}...") # Print last 200 characters
        print(f"TOTAL CHARACTERS: {len(text)} characters")
        return text # Return extracted text
    except Exception as e:
        print(f"Error processing PDF: {str(e)}")
        return None
```

Figure 4.24: PDF extraction code snippet

```python
# Upload PDF file to GridFS and return the URL to access it
@app.route('/api/upload-pdf', methods=['POST'])
def upload_pdf():
    if 'pdf' not in request.files: # Check if 'pdf' part is in the request
        return jsonify({"error": "No pdf part"}), 400

    file = request.files['pdf'] # Get the file from the request
    if file.filename == '': # Check if filename is empty
        return jsonify({"error": "No selected file"}), 400

    try: # Try to process the file
        # Store file in GridFS
        filename = secure_filename(file.filename) # Secure the filename
        file_id = fs.put(
            file,
            filename=filename,
            content_type=file.content_type
        ) # Store the file in GridFS

        # Generate URL to access the PDF
        # Use environment variable for production URL
        base_url = os.environ.get('SERVICE_URL', 'http://localhost:9090')
        pdf_url = f"{base_url}/pdfs/{str(file_id)}"

        return jsonify({"pdfUrl": pdf_url})
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/pdfs/<file_id>')
def serve_pdf(file_id):
    try:
        # Find file in GridFS
        file_data = fs.get(ObjectId(file_id))

        # Create response with proper content type
        response = send_file(
            file_data,
            mimetype='application/pdf',
            as_attachment=False,
            download_name=file_data.filename
        )

        return response
    except Exception as e:
        return jsonify({"error": str(e)}), 404
```

Figure 4.25: PDF Upload code snippets



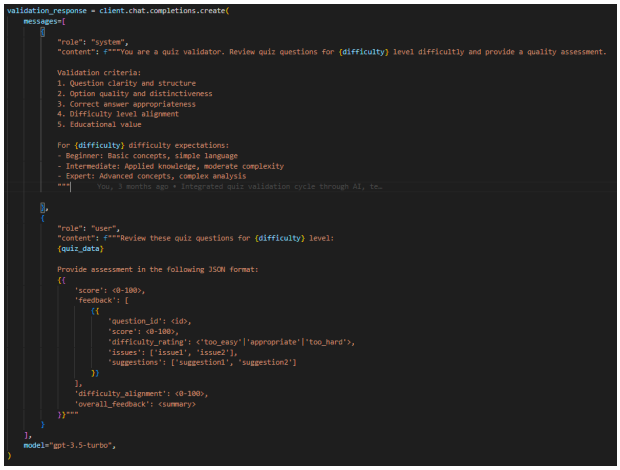


Figure 4.26: Question Validation UI

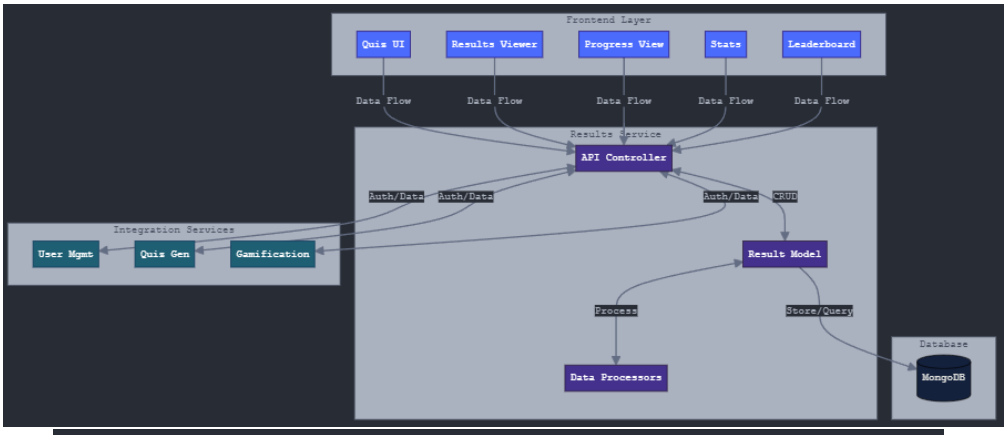
Figure 4.27: Question validation code snippet

Its result is returned to the user, providing them with feedback on the quality of each question generated. Should they wish, users can make manual adjustments to the questions and then submit for another validation. This all works to ensure that a user can create effective study material quickly that caters to their needs. Should the case arise where the generation returned is subpar, they're aware of it. No questions are ever generated for the user without being validated first.

All access to this microservice is done through RESTful endpoints and appropriate authentication is enforced, so users can only generate or access quizzes if they have an account.

Results Tracking Microservice

The Results Tracking Microservice is responsible for recording a user's results on a quiz. They are sent from the frontend back to this microservice for storage, analysed and prepared for visualising on the frontend. This is a crucial element of the platform, and was realised as such shortly after the larger quiz attempt system was employed. It became quickly apparent how much a user could benefit from receiving meaningful feedback as they study, giving them data-driven insight. Additionally, teachers or other students could inspect a student's profile and see the performance.



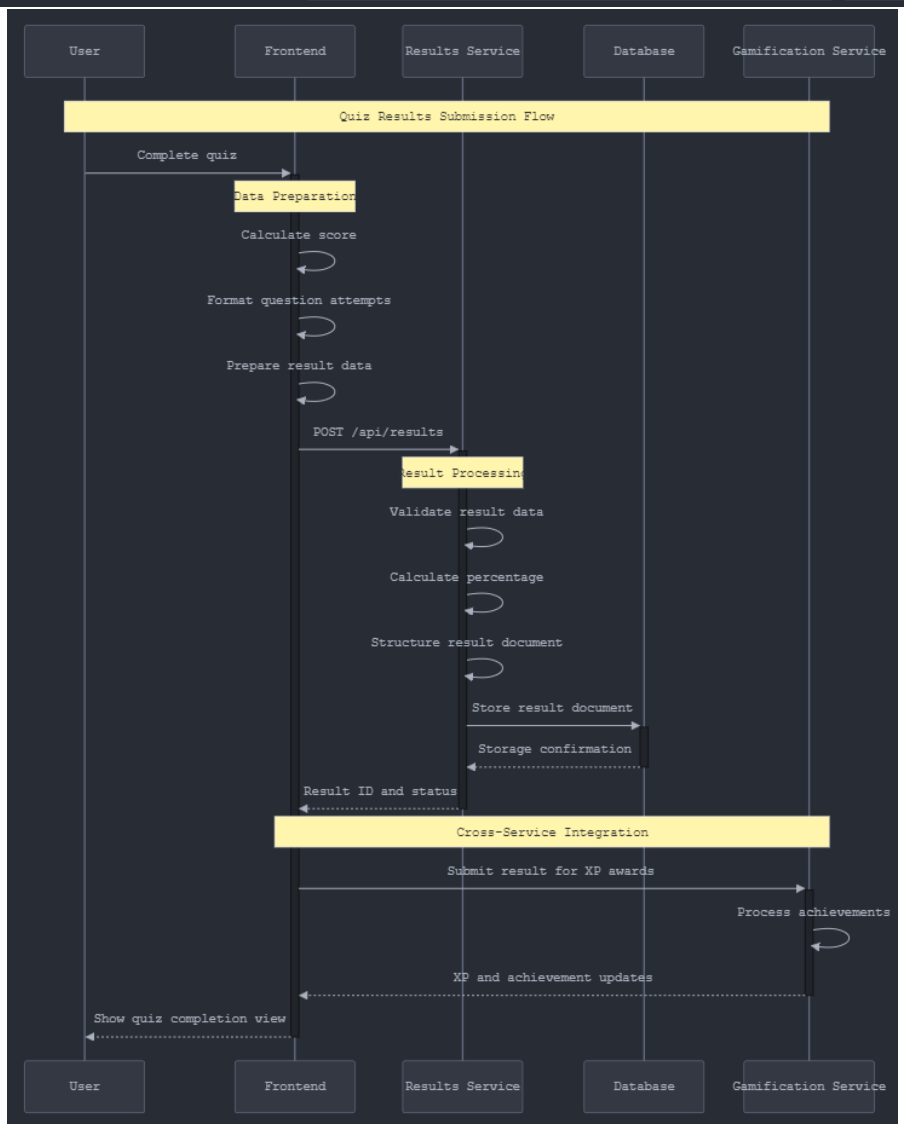


Figure 4.28: Results Microservice architectural diagram

Figure 4.29: Sequence Diagram: Results submission process

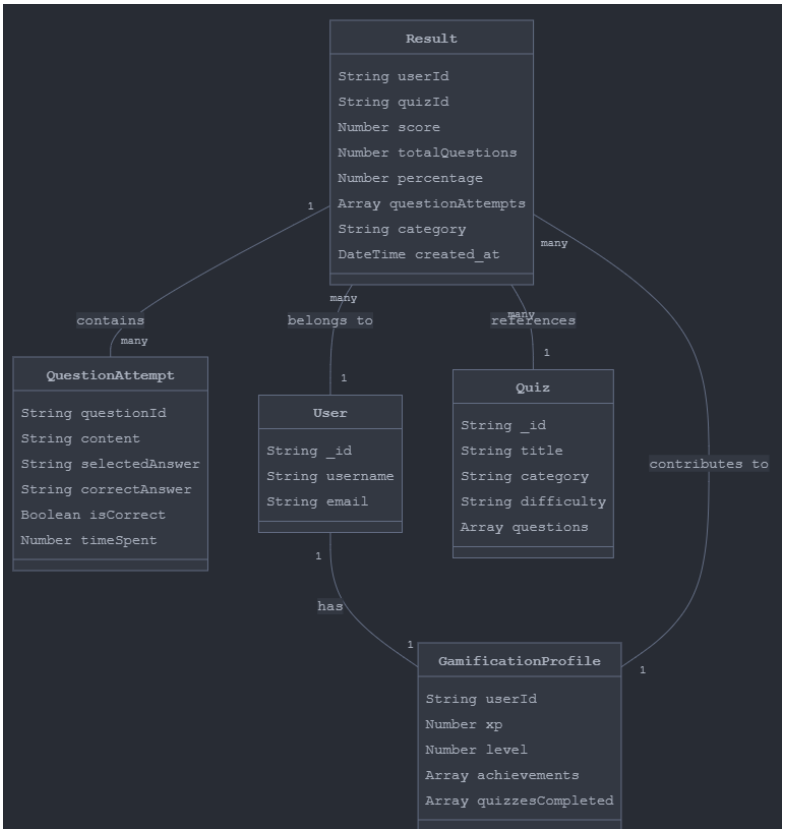


Figure 4.30: Results data model

key components include:

1. Results Storage: Records information about quiz attempts by a user. This includes total questions, percentage correct, total attempts and study category. This data is stored in MongoDB with appropriate IDs. Both the userID of the user who completed the quiz and the quizID are stored.

2. Visualisation Data Preparation: Results are optimised for visualisation in the frontend using D3.js.

Integration with D3.js

Results are returned from the Results Tracking Microservice to produce a number of graphs and charts in the frontend. They are varied and provide the user a number of insights on their learning. Several were tested and adjusted, with many granting better results than others or appearing more visually appealing.

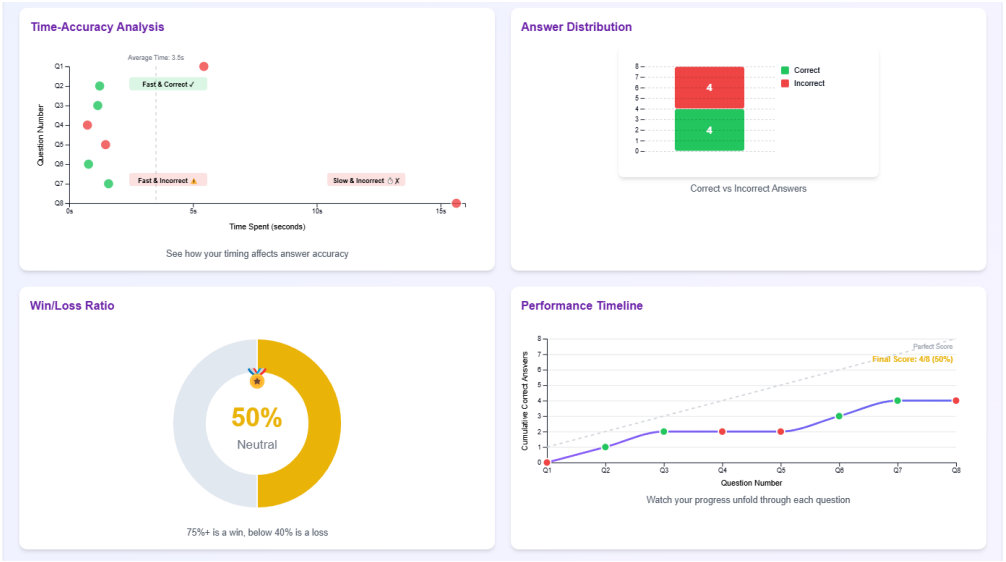
They were broken up into two spaces of functionality :

1. Category Progress Graphing: These are graphs that showcase a user's overall progress across the quizzes they've done, filtered by category of study. These include:

    - Bubble Charts of total quizzes complete per category. These can be clicked to change inspected results.
    - Progress Timelines - A line graph showcasing the high and lows of a user's growth over time.
    - Radar Chart - Four pronged chart showcasing a user's skill across several aspects: Accuracy, Consistency, Improvement and general Completion.
    - Performance Heatmap - A selection of gradient bars that showcase a user's score in quizzes over a weekly basis.

2. Per Quiz Graphing: These are graphs produced as a question is attempted in a quiz. When completed, several graphs are presented to the player to present their results. These include:

    - Performance Timeline - A Line Graph showcasing cumulative correct answers throughout the quiz, along with a dotted grey line for what a perfect score would be.
    - Answer Distribution - A Box Plot that presents correct vs incorrect answers.
    - Win/Loss Ratio - An edited Pie Chart that displays a percentage ratio of success in the quiz depending on the result achieved.
    - Time Accuracy Analysis - A Dot Plot examines your time spent on a question against your accuracy.



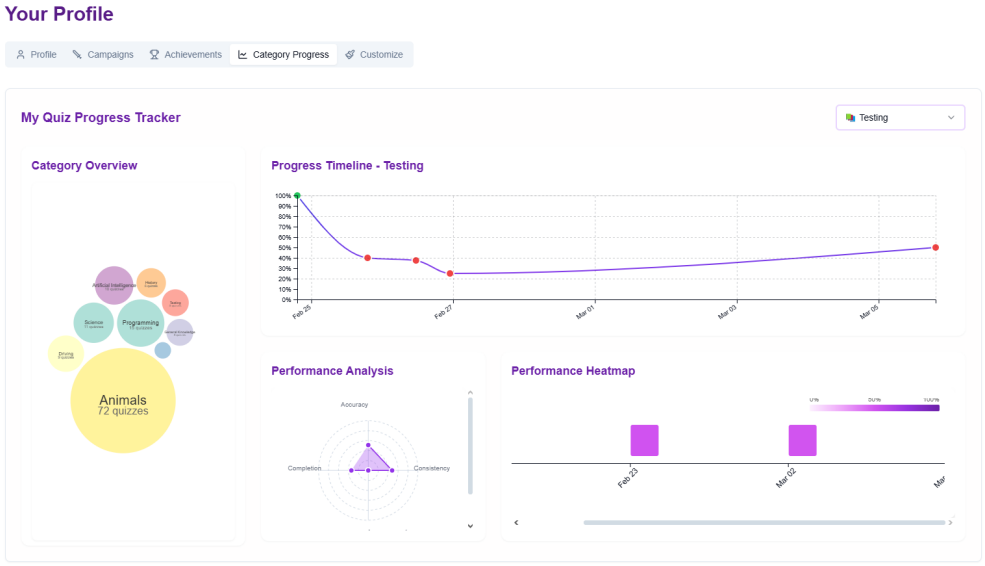


Figure 4.31: Quiz Results data showcased within quiz results

Figure 4.32: Category Progress data showcased on an a player's profile

```python
def createResult(resultData):
    from db import results_collection

    # Validate required fields
    required_fields = ['userId', 'quizId', 'score', 'totalQuestions', 'questionAttempts', 'category']
    for field in required_fields:
        if field not in resultData: # check if all required fields are present in the resultData
            raise ValueError(f"Missing required field: {field}")

    # Calculate percentage
    percentage = (resultData['score'] / resultData['totalQuestions']) * 100

    result = { # create a new result object
        'userId': resultData['userId'],
        'quizId': resultData['quizId'],
        'score': resultData['score'],
        'totalQuestions': resultData['totalQuestions'],
        'percentage': percentage,
        'questionAttempts': resultData['questionAttempts'],
        'category': resultData['category'],
        'created_at': datetime.utcnow()
    }

    try: # insert the result into the database
        response = results_collection.insert_one(result)
        return str(response.inserted_id)
    except Exception as e:
        print(f"Database errorL {str(e)}")
        raise
```

Figure 4.33: Create results code snippet

```python
# Route to get all results for a user in a specific category
@app.route('/api/results/category/<user_id>/<category>', methods=['GET'])
def get_category_results(user_id, category):
    try:
        from db import results_collection
        results = list(results_collection.find({
            'userId': user_id,
            'category': category
        }).sort('created_at', 1))  # Sort by date ascending

        # Convert ObjectId to string for JSON serialization
        for result in results:
            result['_id'] = str(result['_id'])
            if 'percentage' not in result:
                result['percentage'] = (result['score'] / result['totalQuestions']) * 100
            result['percentage'] = float(result['percentage'])

        return jsonify(results)
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Figure 4.34: Category Results code snippet

This microservice also possesses a degree of overlap with the Gamification Microservice for the delivery of the Leaderboard functionality, allowing for the retrieval of each individual user's total quizzes complete for inclusion in the board's

ranking logic.

This microservice communicates briefly with the Quiz Generation service (for quiz metadata retrieval) and the Gamification service (for leaderboard and reward triggering based on performance). It uses RESTful endpoints for quiz results submission, for data retrieval and graph preparation.

Gamification Microservice

The gamification microservice manages all the gaming elements that are developed within the platform to enhance a user's engagement and encourage their motivation to learn. These game mechanics help to reward users for study progress, give them a sense of achievement and maybe give them more consistent study habits.

Many aspects of gamification within study platforms was explored, particularly researching platforms like kahoot and Duolingo for how they operated. To continue the trend of separating concerns between services, each user is given a "Player" model for tracking their gamified statistics that is linked with the user via their userID.

```python
# Alternative method for getting detailed player stats including level, XP, achievements, and category progress
@app.route('/api/users/<user_id>/<username>/stats', methods=['GET'])
def get_player_stats(user_id, username):
    try:
        player = db.gamificationdb.players.find_one({'user_id': user_id})
        if not player:
            print(f"Creating new player data for user_id: {user_id} with username: {username}")
            # Create new player profile if it doesn't exist
            newPlayer = Player(user_id=user_id, username=username).to_dict()
            db.gamificationdb.players.insert_one(newPlayer) # Insert new player into database
            return jsonify(newPlayer), 201 # Return the new player profile, 201 for created status

        # Get player's achievements count
        achievements_count = len(player.get('achievements', []))

        # Get streaks
        streaks = db.gamificationdb.streaks.find_one({'user_id': user_id, 'category': None})
        streak_days = streaks.get('current_streak', 0) if streaks else 0

        # Get quizzes completed and perfect scores from the results database (if available)
        quizzes_completed = player.get('quizzes_completed', 0)
        quizzes_perfect = player.get('perfect_scores', 0)

        # Get category progress
        category_levels = player.get('category_levels', {})
        category_progress = []

        for category, data in category_levels.items():
            category_progress.append({
                'category': category,
                'level': data.get('level', 1),
                'xp': data.get('xp', 0),
                'totalXpRequired': 500 * data.get('level', 1)  # 500 * level for next level
            })

        # Create response object with all player stats
        stats = {
            'level': player.get('current_level', 1),
            'xp': player.get('xp', 0),
            'totalXpRequired': 500 * player.get('current_level', 1),  # 500 * level for next level
            'streakDays': streak_days,
            'quizzesCompleted': quizzes_completed,
            'quizzesPerfect': quizzes_perfect,
            'totalAchievements': achievements_count,
            'categoryProgress': category_progress
        }

        return jsonify(stats), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Figure 4.35: Player stats creation and retrieval code snippet

Here is a full overview of each gamified element this system implemented:

- Experience Points (XP) System: User progress in the app is tracked through an accumulative gain of experience points. This is where the app took its name from. Points are rewarded for completing quizzes, doing daily challenges, earning badges or completing achievements as well as other positive behaviour like maintaining daily streaks. XP gained is pool either into the player's overall XP gained on the platform or into the specific category being studied.
- Levelling System: Accumulated XP is converted into levels. This includes both global levels that indicate overall engagement with the platform and category-specific levels that show expertise in the respective category of study. Each level requires progressively more XP to achieve than the previous.

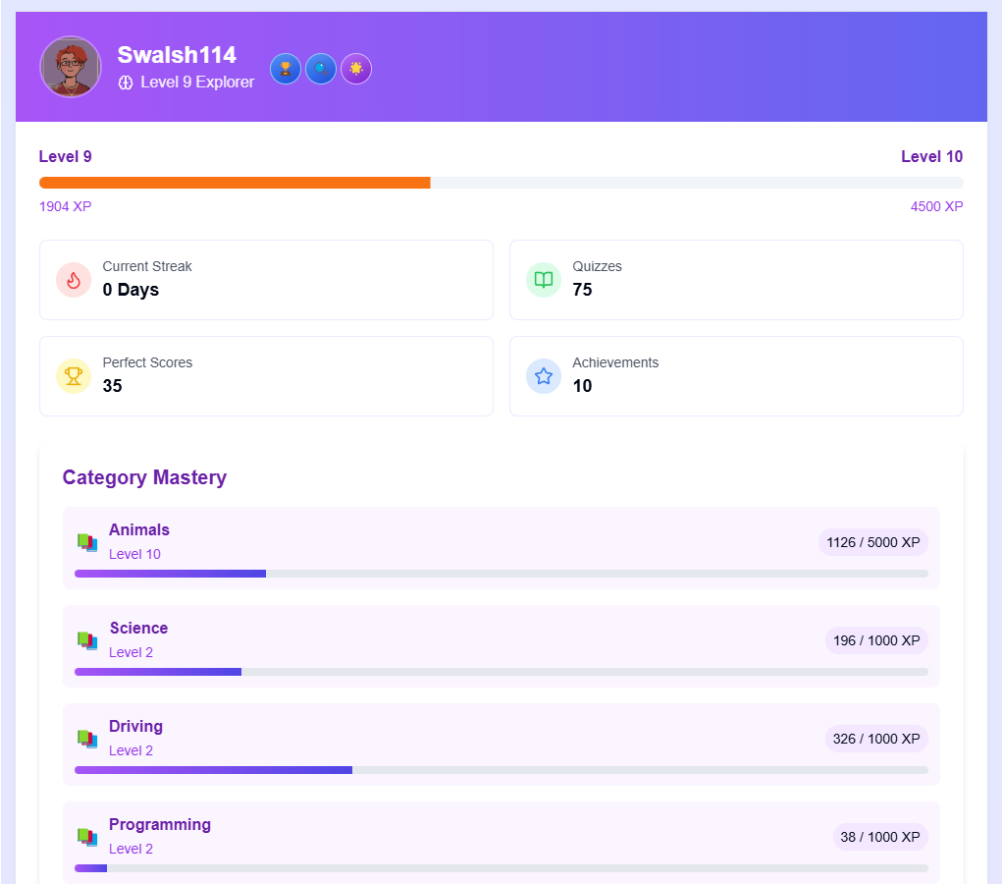


Figure 4.36: Gamified Dashboard UI with player stats

- Achievement System: A fully fledged achievement system is developed. This involves achievement objects that are stored in the database, and user's can unlock them by meeting their specified criteria. A variety of ways are used to check if a user has trigged an achievement milestone or not. They are organised into a series of categories:

- Milestone Achievements - Completing an amount of quizzes, earning an amount of points.

- Performance Achievements - Earning Perfect Scores (80 percent or more correct), speed challenges.

- Engagement Achievements - Daily streaks, multiple subject study.

```python
@app.route('/api/player/<user_id>/achievements', methods=['POST'])
def award_achievement(user_id):
    try:
        data = request.json
        achievement_id = data.get('achievement_id') # Get achievement id from request data

        # Find achievement
        achievementData = db.gamificationdb.achievements.find_one({'achievement_id': achievement_id})
        if not achievementData: # Return error if achievement not found
            return jsonify({'error': 'Achievement not found'}), 404

        # Find player
        playerData = db.gamificationdb.players.find_one({'user_id': user_id})
        if not playerData: # Return error if player not found
            return jsonify({'error': 'Player not found'}), 404

        # Check if player has already earned the achievement
        if achievement_id in playerData.get('achievements', []):
            return jsonify({'error': 'Achievement already earned'}), 400

        # Award achievement to player and xp
        xp_reward = achievementData.get('xp_reward', 0) # Get xp reward from achievement

        # Update player document in db
        db.gamificationdb.players.update_one({'user_id': user_id}, {
            '$push': {'achievements': achievement_id},
            '$inc': {'xp': xp_reward}
        })      You, 2 months ago • Worked on player and achievement endpoints

        # Check for level up
        current_xp = playerData.get('xp', 0) + xp_reward # Get current xp
        current_level = playerData.get('current_level', 1) # Get current level
        level_up = False # Initialize level up flag

        while True: # Loop to check for level up
            next_level_xp = 1000 * (current_level * 0.5)
            if current_xp >= next_level_xp: # If xp is greater than next level xp, level up
                current_level += 1
                level_up = True
            else: # If xp is not enough for level up, break loop
                break

        if level_up: # Update player level if level up
            db.gamificationdb.players.update_one(
                {'user_id': user_id},
                {'$set': {'current_level': current_level}}
            )
```

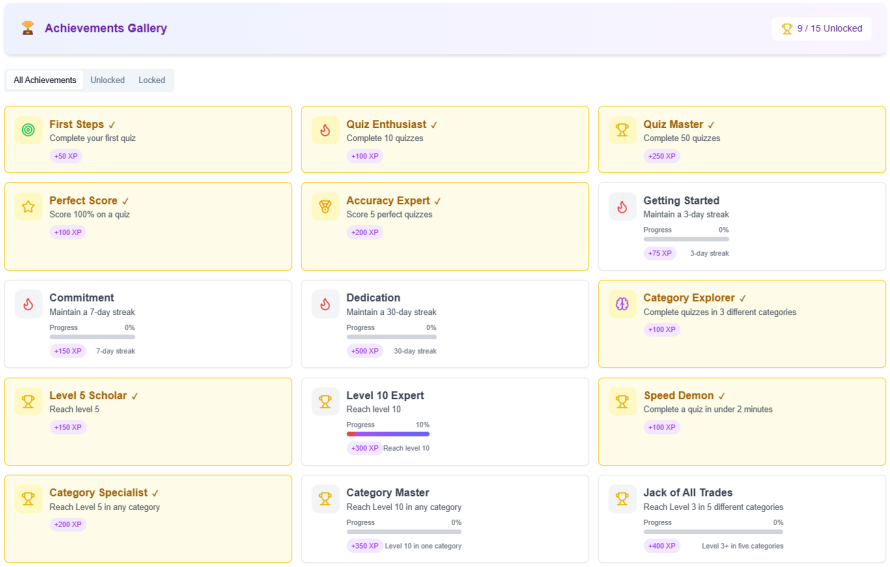


Figure 4.37: Achievement rewarding code snippet

Figure 4.38: Achievements showcase UI within Exper

- Badge System: These are visual awards, sort of like stamps. They are earned when ever an achievement is complete or for select category levels reached. User's can display them on their profile to showcase what significant accomplishments they've made. They serve as a form of status symbol, a player's profile can be customised to showcase whichever ones they prefer.

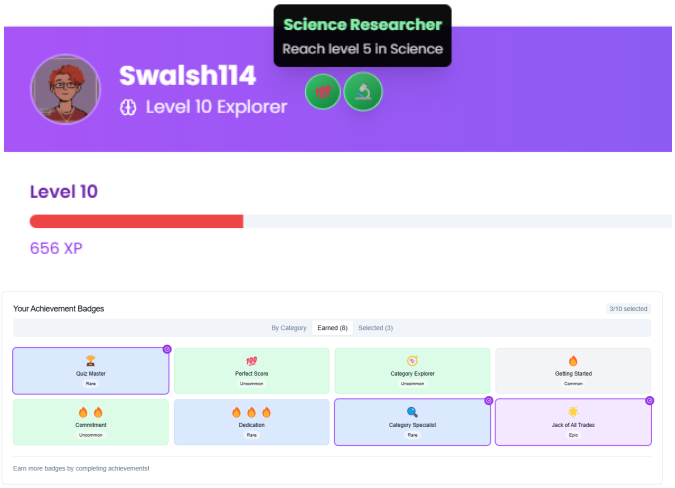


Figure 4.39: Badge Display on a player profile

Figure 4.40: Badge Selection in player settings

Leaderboard System: Maintains rankings across different metrics within the app. Whether that be total quizzes completed, achievements earned, total xp gained or more. This is a classic game mechanic that fosters some friendly competition amongst peers.

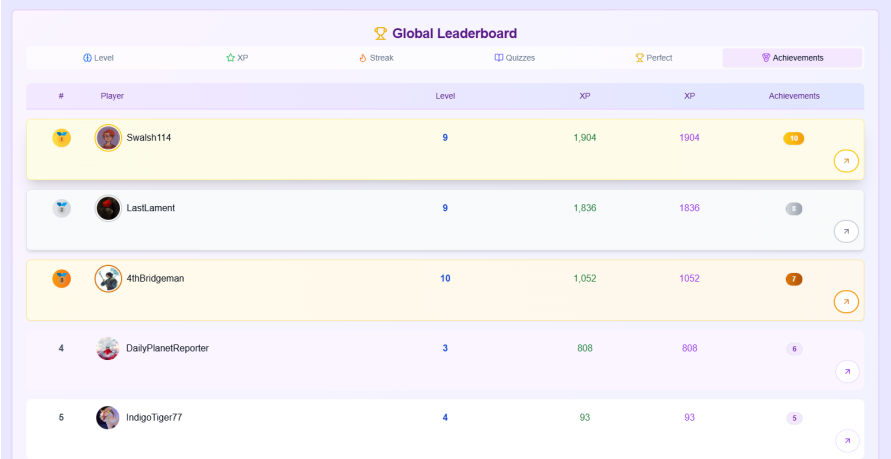


Figure 4.41: Global Leaderboard UI with various players

```python
@app.route('/api/leaderboard', methods=['GET'])
def get_leaderboard():
    """Get all players sorted by various stats for the leaderboard"""
    try:
        # Get all players from the database
        players = list(db.gamificationdb.players.find({}))

        # Prepare player data for leaderboard
        leaderboard_data = []
        for player in players:
            # Convert ObjectId to string for JSON serialization
            player['_id'] = str(player['_id'])

            # Create a leaderboard entry with necessary fields
            leaderboard_entry = {
                '_id': player['_id'],
                'user_id': player['user_id'],
                'username': player.get('username', 'Unknown Player'),
                'level': player.get('current_level', 1),
                'xp': player.get('xp', 0),
                'streakDays': 0,  # Will be populated below
                'quizzesCompleted': player.get('quizzes_completed', 0),
                'quizzesPerfect': player.get('perfect_scores', 0),
                'totalAchievements': len(player.get('achievements', [])),
                'profileImage': player.get('profile_image', None),
                'imageUrl': player.get('image_url', None)
            }

            try:
                user_data = db.userdb.usercollection.find_one({'_id': ObjectId(player['user_id'])})
                if user_data and 'imageUrl' in user_data and not leaderboard_entry['profileImage']:
                    leaderboard_entry['imageUrl'] = user_data['imageUrl']
            except Exception as e:
                app.logger.error(f"Error fetching user details for leaderboard: {e}")

            # Get player's current streak if available
            streak = db.gamificationdb.streaks.find_one({
                'user_id': player['user_id'],
                'category': None  # Get the overall streak, not category-specific
            })
            if streak:
                leaderboard_entry['streakDays'] = streak.get('current_streak', 0)

            leaderboard_data.append(leaderboard_entry)

        return jsonify(leaderboard_data), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Figure 4.42: Leaderboard handling code snippet

The frontend works in tandem with the gamification service in many regards. It allows for visualising the levelling progress and presenting badges to the player or the current leaderboards. A large notification handling system was integrated to visualise to a player when they have completed an achievement or earned a badge.

Similar to Toast, it acts as a layer applied above the page within the layout.tsx file, but its far more visually appealing and intuitive than Toast. On top of that, a fun little Confetti plug-in was implemented that triggers for achievement or badge rewards, notifications or level ups.

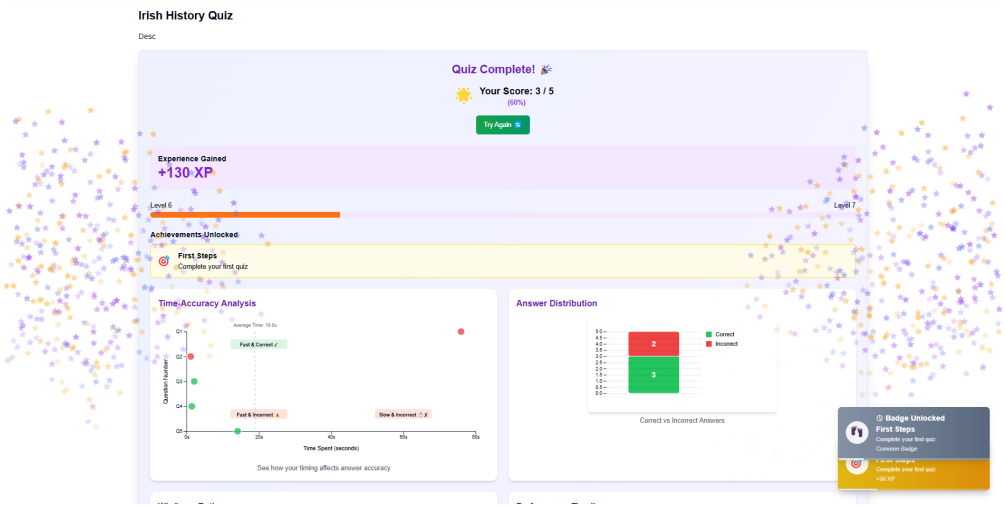


Figure 4.43: An Achievement/Badge notification trigger in Exper (Yes that's confetti!)

There was a desire to develop a campaign system within the platform. This would allow user's to have guided paths for studying within the platform. It was partially implemented but scrapped for time.

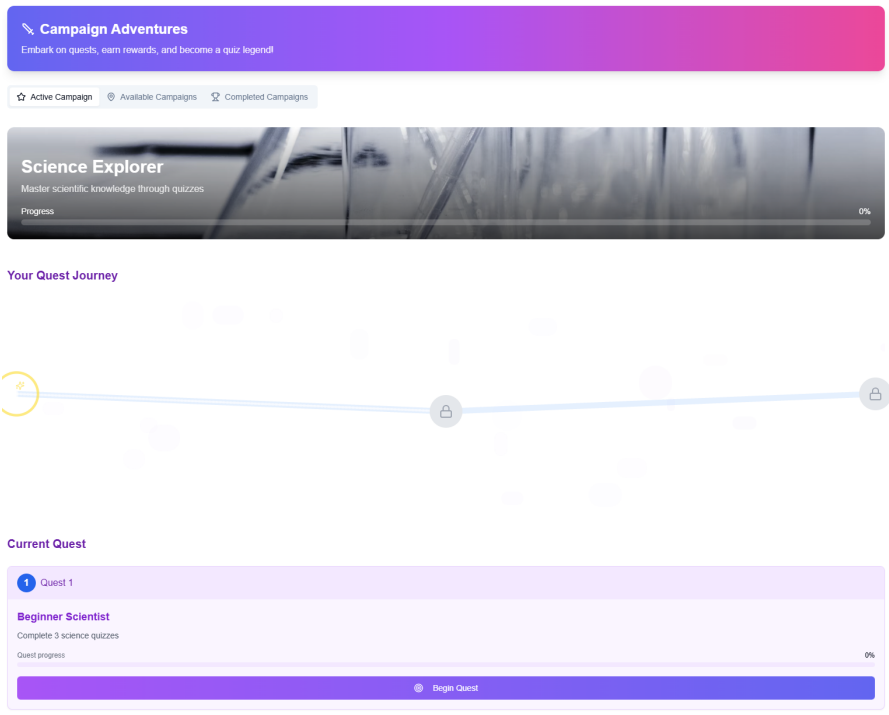


Figure 4.44: Partially implemented campaign system within Exper

The Gamification microservice integrates with the Results Tracking service for select monitoring of results for achievement triggering and interacts with the User Management Service for linking userIDs to Player stats.

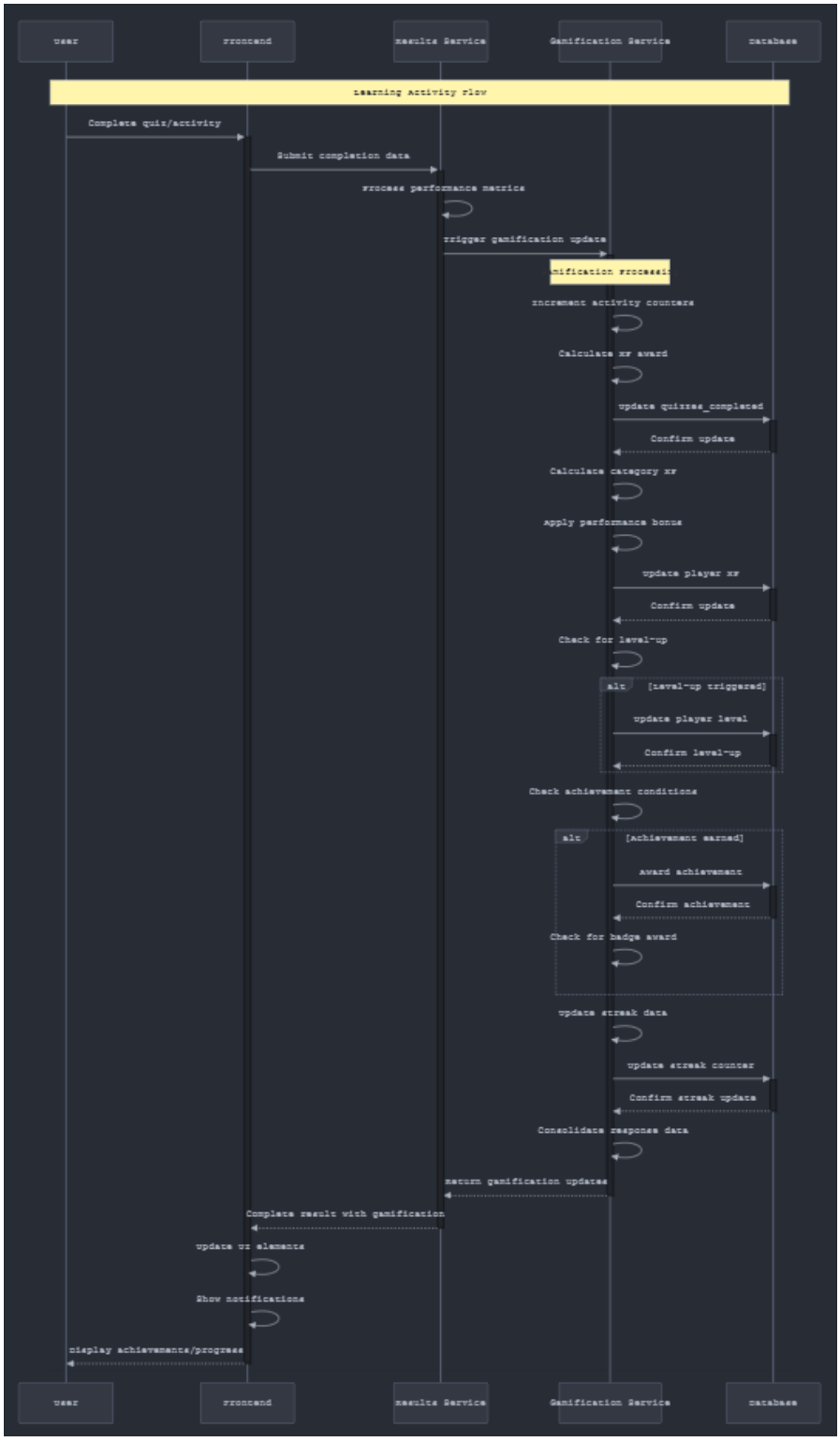


Figure 4.45: Sequence Diagram: Player stats and gamification process

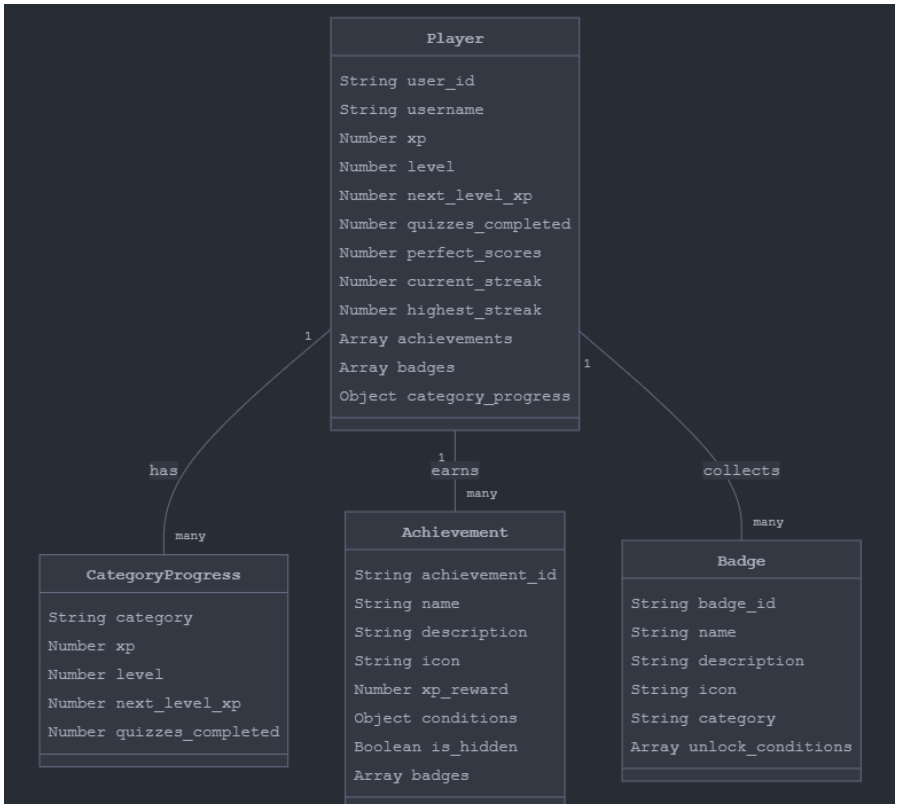


Figure 4.46: Gamified features data model

4.0.3 Security

Security was considered throughout Exper's design. Several measures were taking to adhere to known standards. This includes:

- Authentication: JWT-based authentication with token expiration and refreshing mechanisms. Passwords are hashed and stored using Werkzeug, a python extension so personal user data isn't compromised.
- Authorisation: Much of the platforms functionality across the frontend and within several microservices is locked behind user authorisation, where in the existence of a user must be checked with the User Management Service before a feature can be used. This includes middleware validating permissions for each request made.
- Data Protection: All communication is done through HTTPS encryption.
- API Security: Rate limiting is in place and CORS policies are applied for

communicating with every microservice. Input validation is used in select cases to reduce chances of cyber attacks such as injection attacks.

- Container Security: The use of Docker containers and how they're configured kept permissions in check, and the deployment strategy aids in preventing security breaches.

```
# Configure CORS
CORS(app,
     resources={r"/api/*": {
         "origins": ["http://localhost:3000", "https://exper-frontend-production.up.railway.app", "https://expergle.com"],
         "methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],
         "allow_headers": ["Content-Type", "Authorization", "Accept"],
         "expose_headers": ["Content-Type", "Authorization"],
         "supports_credentials": True,
         "allow_credentials": True,
         "max_age": 120
     }},
     supports_credentials=True)
```

Figure 4.47: Cors Policy handling in a microservice

4.0.4 Deployment

Deployment was explored through a number of avenues. Docker was always involved in the process, but some attempts of deployment involved the use of Kubernetes. Deployment was attempted on Digital Ocean as well as Vercel before settling for Railway.

Railway Deployment

This uses a containerised approach, with each microservice and the Frontend deployed in seperate Docker containers on Railway. This allows for individual changes to be made to one without having to re-deploy the entire system.

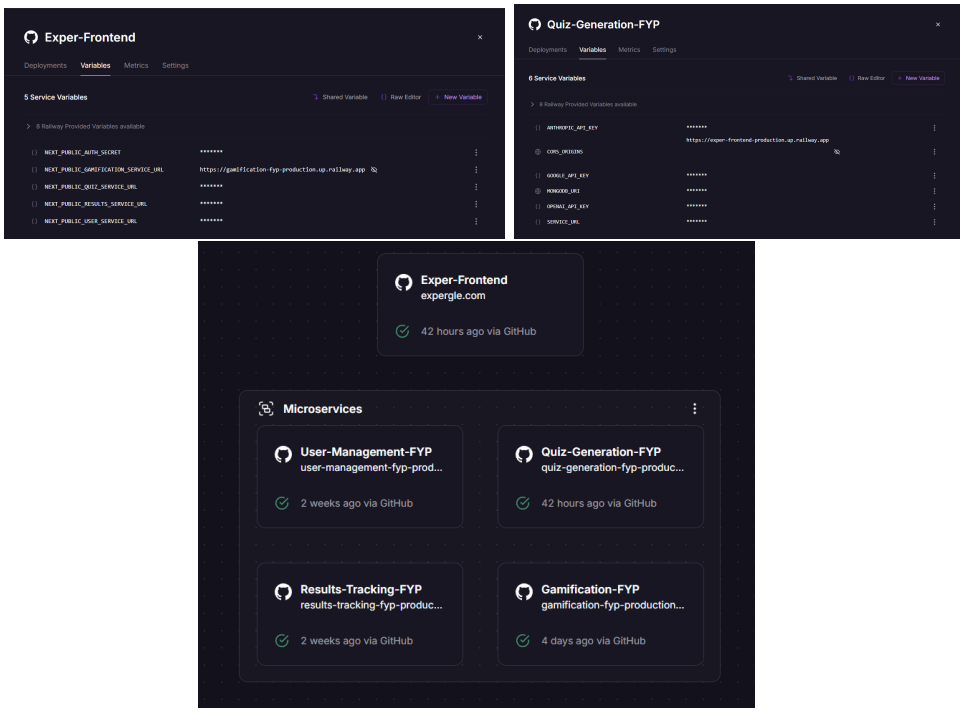
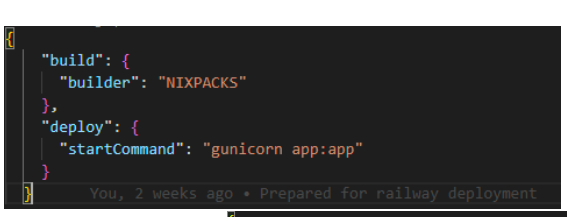


Figure 4.48: Environmental Variable handling for the Frontend

Figure 4.49: Envrionmental Variable handling for the Quiz Generation Service

Figure 4.50: Exper's architecture in Railway, each node is a separate deployment

Each service and the frontend has a configuration file known as railway.json. This specifies build and deployment parameters for Railway to follow when deploying your environment. Most services had the same configuration, however the Frontend container required a slightly different approach for Next.js start ups. Furthermore, The Quiz Generation service required far more resources due to its AI component taking up Railway's average amount of workers given. For seamless communication across all elements of this system, environment variables are appropriate handled for service communication and Generative AI API communication.



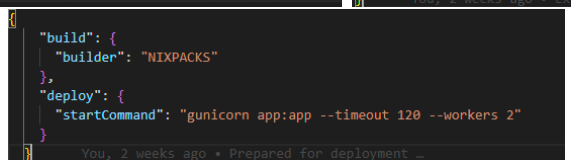


Figure 4.51: Railway configuration for a typical microservice

Figure 4.52: Railway configuration for Next.js Frontend

Figure 4.53: Railway configuration for Quiz Generation microservice

Chapter 5

System Evaluation

This chapter focuses on critically evaluating the Exper Gamified Learning Environment that was developed against its original objectives. Strengths of the system as well as weaknesses will be highlighted. It will be examined in terms of quality of success and reflected on for the technologies used and approaches taken.

5.0.1 Evaluation against Objectives

This section will evaluate how well the GLE met each of the key objectives outlined in the introduction.

Personalised Learning Platform

The primary goal was to create a customisable personalised learning platform that adapts to individual student needs. Overall, this implementation delivered:

- Custom Quiz Generation: Users can create quizzes quickly and on the fly from their own study materials, the option is implemented to either manually craft the quiz question or use AI generation from notes or PDFs. This delivered on the initial objective.

- Multiple AI Models: OpenAI's GPT, Anthropic's Claude and Google's Gemini are all integrated to allow for choice and comparison. They each provide differences in generation capabilities, with each showing different strengths in question creation.

- Customisation Parameters: Users are able to specify difficulty levels, types of questions, number of questions as well as randomise with question pools. This gives them control over their study experience.

- Subject Categorisation: The use of subject categories allows students to hone in on specific areas of study and track progress within the particular subject.

Gamification in Education

The Gamification elements were widely implemented across the application in multiple variants to a successful degree. This delivered on:

- Experience System: The levelling system provides the user with continuous progression as they learn, in hopes of increasing motivation to learn to further their levels and achieve milestones.
- Achievement System: 28 Unique achievements are built within the system and can be earned, with dynamic notifications. They vary in categories (study streaks, quiz completion, category mastery to name a few), offering users a variety of clear goals to chase and show off.
- Badges: Visual rewards displayable on a user's profile. Gained from engagement on the platform and study within categories, further aid to keep engagement.
- Leaderboards: Both global and category specific stat rankings that foster friendly competition amongst users.

Quiz Generation Implementation

Quiz generation was a core goal of this platform, aiming to deliver a system for creating study quizzes with a high degree of control over the degree and format. While the degree of control could always evolve and be pushed further, its capabilities can be measured quite apt given the time restraints and other features at play within the platform as well.

This implementation delivered:

- manual Creation Tools: Functionality for users or educators to craft specific quiz questions manually, choosing between question types, titles, categories etc.
- AI Integration: Having ChatGPT, Claude and Gemini all integrated seamlessly through the use of prompt engineering techniques delivers on an ability to craft quizzes quickly and efficiently, whether from users notes, PDFs or just a simple guiding sentence. Furthermore, these AIs can be used for validation of questions for quality, regardless of whether they were generated or written.

- Note Processing: The System delivers on handling student notes effectively, handling pasted notes in the AI process and uploaded PDFs are parsed and extracted of text to be used within the quiz generation cycle.

Comparative testing between each AI model is a fascinating finding. Overall, Claude produced slightly more rigorous questions in terms of difficulty, adhering to the difficulty tiers fed into it more effectively than the others. By comparison, Gemini struggles far greater with adhering to the JSON layout engineered in its prompt. 1/10 results from Gemini are shown to break format, but its ability to search could be of benefit. Lastly, ChatGPT excelled at generating more creative questions, which may test you in unique and interesting ways where the other models won't.

Intuitive and Responsive Frontend

The Next.js frontend developed achieves:

- Cross-device Compatibility: Seamless functionality is present across desktop and mobile devices, largely thanks to the use of Next.js and its dynamic components.
- Visual Appeal: The interfaces aesthetic was developed to be intuitive and pleasing for studying within. The use of Shadcn-ui along with Tailwind CSS played a major role in helping design this modern clean look.
- Middleware and services: The Frontend handles lots of middleware functionality for seamless interaction and implementation of each of the backend microservice functionalities.
- Component construction: Extensively developed frontend components work with the microservices to deliver much of the backend functionality, or to compliment it. Examples include the D3.js performance graphs, dynamic achievement notifications or the question navigation and flagging during a quiz attempt.

Microservices Architecture

The architecture stuck to its microservice philosophy, with select imports such as D3.js to complement a services implementation.

- Four distinct services: User Management, Quiz Generation, Results Tracking and Gamification. These were set out at the start and while the thought of additional services was considered, never was one of the core four abandoned.

- Development philosophy: The methodology was successfully adhered to, developing each service in separate sprints throughout the two semesters.
- RESTful Mindset: All services communicate with the frontend efficiently through the use of REST APIs.
- Containerisation: Docker containers help maintain the aim of a modular scalable environment. Each operates independently and are deployed separately with routing handled through environment variable on Railway.

From start to finish, the microservice architecture aided in breaking up the development workflow. While one could argue that challenges arose in regards to managing the relationship between multiple services and how to handle them, the findings within developing this platform would have to disagree. The separation did wonders for workflow, modularity and allowed focus on one aspect of functionality at a time.

Limitations and Opportunities

Technical Limitations:

- In many ways, AI response times, page load times could be improved within the platform to be more efficient. Aspirations are present to improve this, but lack of time ultimately became a hindrance on it.

Teaching limitations:

- During the Christmas seminar, there was considerable concerns over the possible quality of the material generated for study. Multiple measures have been taken to counteract this however, such as the implementation of multiple generative AI providers into the system and a full validation process. While these may not be perfect at improving the Pedagogical quality, they act well to work towards it.
- The gamification this platform focuses on could be seen as a distraction within the learning process. There is a considerable fear that it could over power the education. This is a risk the platform accepts and with its development, aims to show how education and gamification can work together if handled correctly.
- The platform largely focuses on the individual user experience, and has little implementation for team based learning, Additionally, there is also something to be said in regards to developing more features to better involve

teachers within the Exper platform, similar to what is seen on platforms like Google Classroom. These are most definitely limitations that are eager to be remedied. Integration with educational platforms like Moodle would go a long way with developing student regulation and monitoring systems.

Future opportunities:

- A feature first approach was taken throughout this project's development with testing during the coding process, there are significant opportunities to expand into a full testing approach for this platform to assure its quality.
- There were aspirations for the possible implementation of a Recommendation service, where AI would use your quiz results to aid in recommending more quiz opportunities to you. Similarly to the Campaign system mentioned in Chapter 4, these fell out of scope due to their considerable size and time necessary to develop. There is great opportunities to implement these in the future.
- As mentioned previously, Kubernetes was attempted to be implemented for container orchestration and scaling, but fell out of scope. Should the opportunity arise to continue development on this process, Kubernetes integration would be one of the first on the menu to be handled.
- From a functionality standpoint, the platforms currently only handles 1 in 4 and Multiple choice questions. There are many more question types that could be implemented whether that be audio files, order sorting questions or fill in the blanks as a couple examples.
- The Results microservice collects a sizeable amount of player data, but there is an opportunity to expand this to more analytics that would grant greater study insights.

5.0.2 Reflection

Exper, the Gamified Learning Environment proved to be a challenge to develop this modern educational AI-integrated platform, but equally rewarding. The microservices architecture proved to be excellent at providing modularity to the codebase and eased management. It will also support well for future scaling or additions like Kubernetes should they be pursued. The decision to integrate numerous generative AI providers into the platform proved valuable. It not only highlights the differences they hold in generating quiz questions, but also lessens any claims that the platform is reliant on any one model. With this, it stands on its own two legs. While it did increase complexity, it also provided redundancy.

The gamification elements were successfully implemented widely across the board. The aim was to implement them into almost every aspect of the platform, and that was achieved. They hopefully act to encourage engagement in the platform. There are however, considerations to be made regarding the balance between gamified features and the education to ensure that they serve the study experience and not fight against it for a user's attention.

The other successful aspect was the personalisation capabilities presented in the platform that allow a student to create custom study materials from their own notes. As stated in the introduction, the aim here was to bridge a fundamental gap that existing educational technologies have. Reflecting on that, this platform provides value that generic study platforms don't have, filling its own niche.

Overall, the GLE met its core objectives while sacrificing in select spots. There are several avenues for future additions and enhancements. The modular architecture translates well for bringing these changes to the platform without requiring a whole system redesign.

Chapter 6

Conclusion

6.0.1 Summary

Throughout this dissertation, the methodology, design, development and evaluation of Exper - The Gamified Learning Platform has been detailed. As stated previously, the aim was to address the lack of engaging study techniques outside of the traditional classroom with our increasingly online world. It was determined that the lack of motivation, interactivity and actionable insightful results were chief in limiting today's students from reaching their learning goals. The main objective of Exper was to remedy this by creating a highly personalised, customisable learning platform that'll enhance the out-of-class study experience. It would do this through interactive quizzes, utilising gamification, performance tracking and AI integration to provide student's with a flexible platform to study and improve at their own pace.

Technically, it centered on a microservices approach to ensure a scalable modular codebase that would keep development segregated and manageable. Key objectives in the platform involved implementing interactive quiz generation using either manually or Generative AI based creation, utilising a user's own material like study notes or PDFs with difficulty adapting on top. This was all possible thanks to prompt engineering allowing the delivery of this personalised process. Lastly, the integration of gamification elements across the platform such as levelling and stats tracking, achievements and leaderboards were crucial to boost motivation and engagement.

As the System Evaluation chapter explored, the GLE in many ways met its objectives, with select elements dropped for the sake of time or focusing on more core functionality.

6.0.2 Final Statement

In conclusion, Exper's development was an aim at providing an extra step towards more engaging and effective study tools in our online world. It successfully integrated modern technologies such as microservices, gamification principles and numerous generative AIs, this has hopefully demonstrated just how many opportunities there are to expand and improve education through the use of technologies and software development. Exper's modular architecture gives it a easy avenue for future growth and adaptation. This project shows that there is a place for the cross section of AI, gamification and analysis within software and technology development to bring students a more enjoyable, accessible learning experience.

6.1 Appendices

6.1.1 GitHub Repository and Jira

https://github.com/Exper
https://ExperJira.com

6.1.2 Screencast Demonstration

https://youtube/FYPScreencast

6.1.3 Visit the deployed site directly

https://Exper.com

Figure 6.1: Exper QR Code to the deployed site

Bibliography

[1] Devanshu Sawarkar, Pratham Agrawal, Rohit Kumar Mandal, Latika Pinjarkar, Poorva Agrawal, and Gagandeep Kaur. The Impact of Gamification Across Different Educational Contexts. In *2024 4th International Conference on Innovative Practices in Technology and Management (ICIPTM)*, pages 1–7, Noida, India, February 2024. IEEE.

[2] Khant Hmue, Myat Pwint Phyu, and Aye Myat Myat Paing. Microservices vs Monolith: A Comparative Analysis and Problem-Solving Approach in Web Development Area. In *2024 5th International Conference on Advanced Information Technologies (ICAIT)*, pages 1–5, Yangon, Myanmar, November 2024. IEEE.

[3] Jose Haro. 2022.

[4] Kunal Sharma, Puneet, Deepika, Pardeep Kumar Jindal, and Preeti Sharma. Empowering Learning: An Integrated Approach to Enhance Education. In *2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT)*, pages 1–6, Greater Noida, India, August 2024. IEEE.

[5] Ivan Silva, Andres Wong, Benito Auria, Dick Zambrano, and Vanessa Echeverria. Gamification in Engineering Education: Exploring Students' Performance, Motivation, and Engagement. In *2022 IEEE Sixth Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6, Quito, Ecuador, October 2022. IEEE.

[6] Jason Tong, Ricky Rivaldo Jikson, and Alexander Agung Santoso Gunawan. Comparative Performance Analysis of Javascript Frontend Web Frameworks. In *2023 3rd International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS)*, pages 81–86, Yogyakarta, Indonesia, August 2023. IEEE.

[7] Lukas Frank, Fabian Herth, Paul Stuwe, Marco Klaiber, Felix Gerschner, and Andreas Theissler. Leveraging GenAI for an Intelligent Tutoring System for R: A Quantitative Evaluation of Large Language Models. In *2024 IEEE*

Global Engineering Education Conference (EDUCON), pages 1–9, Kos Island, Greece, May 2024. IEEE.

[8] Mike Bostock. D3.js - data-driven documents, 2011. A JavaScript library for visualizing data with HTML, SVG, and CSS.

[9] Fatima Vapiwala and Deepika Pandita. Leveraging Gamified Learning Management Systems to Enhance E-Learning Outcomes. In *2024 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 535–540, Sakhir, Bahrain, November 2024. IEEE.

[10] Samuel Kamunya, Elizaphan Maina, and Robert Oboko. A Gamification Model For E-Learning Platforms. In *2019 IST-Africa Week Conference (IST-Africa)*, pages 1–9, Nairobi, Kenya, May 2019. IEEE.

[11] Ana Vrcelj, Natasa Hoic-Bozic, and Martina Holenko Dlab. Using Digital Tools for Gamification in Schools. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 848–852, Opatija, Croatia, September 2021. IEEE.

[12] Mien May Chong, Preethi Subramanian, Mary Ting, and Lau Joe Ying. An Investigation into the Perception of Gen Z Students in Higher Education towards Gamification Techniques in Learning Computing Subjects. In *2024 International Visualization, Informatics and Technology Conference (IVIT)*, pages 165–170, Kuala Lumpur, Malaysia, August 2024. IEEE.

[13] Chiang Liang Kok, Yit Yan Koh, Chee Kit Ho, Tee Hui Teo, and Charles Lee. Enhancing Learning: Gamification and Immersive Experiences with AI. In *TENCON 2024 - 2024 IEEE Region 10 Conference (TENCON)*, pages 1853–1856, Singapore, Singapore, December 2024. IEEE.

[14] Magna Guerrero Celis, Soraya Yrigoyen Fajardo, Giovanna Vassallo Sambuceti, and Rossana Barros. Higher education student's attitude towards using gamification and its relation with achievement. In *2023 IEEE 3rd International Conference on Advanced Learning Technologies on Education & Research (ICALTER)*, pages 1–4, Chiclayo, Peru, December 2023. IEEE.

[15] Gunjan Pathak and Monika Singh. A Review of Cloud Microservices Architecture for Modern Applications. In *2023 World Conference on Communication & Computing (WCONF)*, pages 1–7, RAIPUR, India, July 2023. IEEE.

[16] Elvisa Gashi, Dhuratë Hyseni, Isak Shabani, and Betim Çiço. The advantages of Micro-Frontend architecture for developing web application. In *2024*

13th Mediterranean Conference on Embedded Computing (MECO), pages 1–5, Budva, Montenegro, June 2024. IEEE.

[17] Benymol Jose and Sajimon Abraham. Exploring the merits of nosql: A study based on mongodb. In *2017 International Conference on Networks & Advances in Computational Technologies (NetACT)*, pages 266–271, Thiruvanthapuram, India, July 2017. IEEE.

[18] Laxmisha Rai, Chunrao Deng, and Fasheng Liu. Developing Massive Open Online Course Style Assessments using Generative AI Tools. In *2023 IEEE 6th International Conference on Electronic Information and Communication Technology (ICEICT)*, pages 1292–1294, Qingdao, China, July 2023. IEEE.

[19] Pragya Singh, Nidhi Phutela, Priya Grover, Deepti Sinha, and Sachin Sinha. Student's Perception of Chat GPT. In *2023 International Conference on Electrical, Communication and Computer Engineering (ICECCE)*, pages 1–6, Dubai, United Arab Emirates, December 2023. IEEE.

[20] M. Jayaram, Yogesh Bhutkar, Indhra Lochan Kumar Bojjanapalli, Giri Yeshwanth, and Burri Yashwanth Reddy. Beyond Automation: AI-Driven Project Management with OpenAI and Prompt Engineering. In *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*, pages 1–6, Sydney, Australia, July 2024. IEEE.

[21] Fan Bao and Jia Chen. Visual framework for big data in d3.js. In *2014 IEEE Workshop on Electronics, Computer and Applications*, pages 47–50, Ottawa, ON, Canada, May 2014. IEEE.

[22] An Bing and Zhu Li-Gu. Film Big Data Visualization Based on D3.js. In *2020 International Conference on Big Data and Social Sciences (ICBDSS)*, pages 50–53, Xi'an, China, August 2020. IEEE.

[23] Tianbo Lu, Peng Zhang, and Huiyang Li. Practice Teaching Reform of Discrete Mathematics Model based on D3.js. In *2019 14th International Conference on Computer Science & Education (ICCSE)*, pages 379–384, Toronto, ON, Canada, August 2019. IEEE.

[24] Qidong Li and Lichen Zhang. Design and Implementation of Online Education Platform with Microservice Architecture. In *2024 IEEE 15th International Conference on Software Engineering and Service Science (ICSESS)*, pages 173–177, Changsha, China, September 2024. IEEE.

[25] Shreyas Agrawal and Dhawan Singh. Study Containerization Technologies like Docker and Kubernetes and their Role in Modern Cloud Deployments.

In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, pages 1–5, Pune, India, April 2024. IEEE.

[26] Vini Kanvar, Ridhi Jain, and Srikanth Tamilselvam. Handling Communication via APIs for Microservices. In *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 37–42, Melbourne, Australia, May 2023. IEEE.

[27] Ar Anil Yasin and Asad Abbas. Role of gamification in Engineering Education: A systematic literature review. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, pages 210–213, Vienna, Austria, April 2021. IEEE.