EduGenius
Software Architecture Document

Version 1.0

Team Members:

PERERA S.A.I.M.	210471F
PRABASHWARA D.G.H.	219483T
PRANAVAN S.	210491P

Group ID: 21
Project ID: 22
Mentor: Dr. Chathuranga Hettiarachchi

Revision History

Date	Version	Description	Author
14/07/2024	1.0	Initial Document	Group 21

Table of Contents

Software Architecture Document

1.Introduction

1.1. Purpose

This Software Architecture Document (SAD) provides a comprehensive architectural overview of the EduGenius system, using a number of different architectural views to depict different aspects of the system. The EduGenius application is a mathematics tutoring platform, and this document aims to capture and convey the significant architectural decisions which have been made during its development highlighting how different components interact, the flow of data, and the underlying infrastructure.

1.2. Scope

This Software Architecture Document (SAD) applies to the entire EduGenius mathematics tutoring application. The development team in charge of putting the application into practice is the intended audience of the System Architecture Document. It provides up the rules, patterns, and design concepts that must be adhered to throughout the design,development and deployment processes. This outlines the system architectural components, interfaces and dependencies, which helps to ensure a logical and well-organized approach to developing the application.

1.3. Definitions, Acronyms, and Abbreviations

- **LLMs** : Large language Models
- **RAG** : Retrieval-Augmented Generation
- **OTP** (One time password): A security measure commonly used in authentication systems to provide a temporary code for accessing a service or account. In the context of this document OTP refers to mobile OTPs: sending a One Time Password Via SMS to a Mobile Phone.
- **REQ**: Requirement
- **USB** : Universal Serial Bus
- **API** : Application Programming Interface
- **SDK** : Software Development Kit
- **HTTP/S** : HyperText Transfer Protocol
- **OCR:** Object Character Recognition
- **O/L:** General Certificate of Education Ordinary Level Examination
- **UI:** User Interface
- **MTTR:** Mean Time To Repair
- **MTBF:** Mean Time Between Failures
- **bugs/KLOC:** Defects per thousand lines of code

1.4. References

The references are added under the reference section of supporting information (section 11)

1.5. Overview

This document provides a comprehensive view of the architecture of the EduGenius Mathematics Tutoring Application. It is organized into several main sections, each addressing a critical aspect of the software architecture:

1. **Introduction:** This section outlines the purpose, scope, definitions, acronyms, abbreviations, references, and an overview of the Software Architecture Document.
2. **Architectural Representation:** Describes the representation and perspectives used to depict the architecture.
3. **Architectural Goals and Constraints:** Details the software requirements and objectives that significantly impact the architecture, including safety, security, privacy, and use of off-the-shelf products.
4. **Use-Case View:** Presents the use-case model and how the architecture supports the significant use cases. This includes detailed use-case realizations.
5. **Logical View:** Provides an overview of the logical architecture, including the key design packages and their relationships.
6. **Process View:** Describes the process architecture, including the key processes and their interactions.
7. **Deployment View:** Gives an idea about how the system is deployed in actual hardware.
8. **Implementation View:** Offers an overview of the implementation model, detailing the decomposition of the software into layers and subsystems, along with any architecturally significant components.
9. **Size and Performance:** Discusses the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.
10. **Quality:** Explains how the software architecture contributes to the system's capabilities other than functionality, such as extensibility, reliability, portability, safety, security, and privacy.

2. Architectural Representation

EduGenius will utilize microservice architecture to implement each functionality. This document gives a detailed look at the architecture of the EduGenius math tutoring platform using the “4+1” architectural view model with RUP conventions

1. **Logical view:** This shows the structure and functions of the system that end-users will see. It Focuses on the logical components like classes, objects, and subsystems.
2. **Process view:** This explains how the system works dynamically, including its behavior during runtime, how it handles multiple tasks at once, and how scalable it is. It shows how different processes within the system interact.
3. **Deployment view:** This focuses on the physical layout of the system, showing how software components are mapped onto hardware when deploying the system. Further it

gives the information how workload is distributed among CPU cores.

4. **Implementation view:** This provides an overview of software components, layers, and subsystems, which is very useful during development.

5. **Use-case view:** This demonstrates how users will interact with the system and how different objects and processes will work together. It includes use-cases to illustrate the system's key functionalities

3. Architectural Goals and Constraints

3.1. Architectural Goals

1. **Enhanced and Personalized Learning Experience**
    - Provide dynamic and contextually relevant guidance and explanations using advanced educational technologies like Retrieval-Augmented Generation (RAG).
    - Deliver real-time feedback and performance tracking to support effective learning.

2. **High Availability and Reliability**
    - Ensure the application is available 90% of the time with minimal downtime.
    - Aim for a Mean Time Between Failures (MTBF) of at least 3,000 hours and a Mean Time To Repair (MTTR) of less than 7 hours.

3. **Security and Privacy**
    - Comply with data protection regulations such as GDPR, COPPA, and FERPA.
    - Implement robust security mechanisms for user identity verification, secure data transmission, and protection against unauthorized access.

4. **Scalability and Performance**
    - Support up to 1000 concurrent users without significant degradation in performance.
    - Maintain core tutoring features during high load conditions or partial system failures.

5. **Modular and Maintainable Design**
    - Follow a modular design approach for easier maintenance and scalability.
    - Ensure code consistency, readability, and adherence to established coding standards.

6. **User-Friendly Interface**
    - Provide a consistent and simple user interface for easy navigation and use.
    - Ensure that a normal user can learn to navigate and use the primary functions within 30 minutes of initial use.

3.2. Architectural Constraints

1. **Regulatory Policies**
    - The application must comply with the General Data Protection Regulation (GDPR) and other regional data protection regulations.

2. **Security Considerations**
    - Implement user identity verification through authentication links sent to users' email addresses upon registration
    - Ensure secure data transmission and storage.

3. **Design Conventions**
    - Following a modular design approach with individual modules for authentication and context generation
    - Use snake_case for naming variables, functions, classes, and files

4. **Communication Protocols**
    - Use HTTPS protocol for secure web page viewing.
    - Use WebSockets for real-time, bi-directional instant messaging in tutoring sessions.

5. **Development Tools and Environment**
    - Use Python for backend development and Streamlit for building the web application
    - Employ Git for version control, GitHub for the code repository, and Docker for containerization
    - Follow a microservices architecture to ensure scalability, maintainability, and flexibility.

6. **External Dependencies**
    - Use Gemini APIs as the Large Language Model (LLM) for the project.
    - Integrate Wolfram Alpha Show Steps API for math solving.
    - Use vision models for image recognition.
    - Integrate Langfuse for LLM output tracing and user feedback evaluation.
    - Use Brevo as an SMTP server for email notifications.
    - Integrate Stripe as payment gateway.

4. Use-Case View

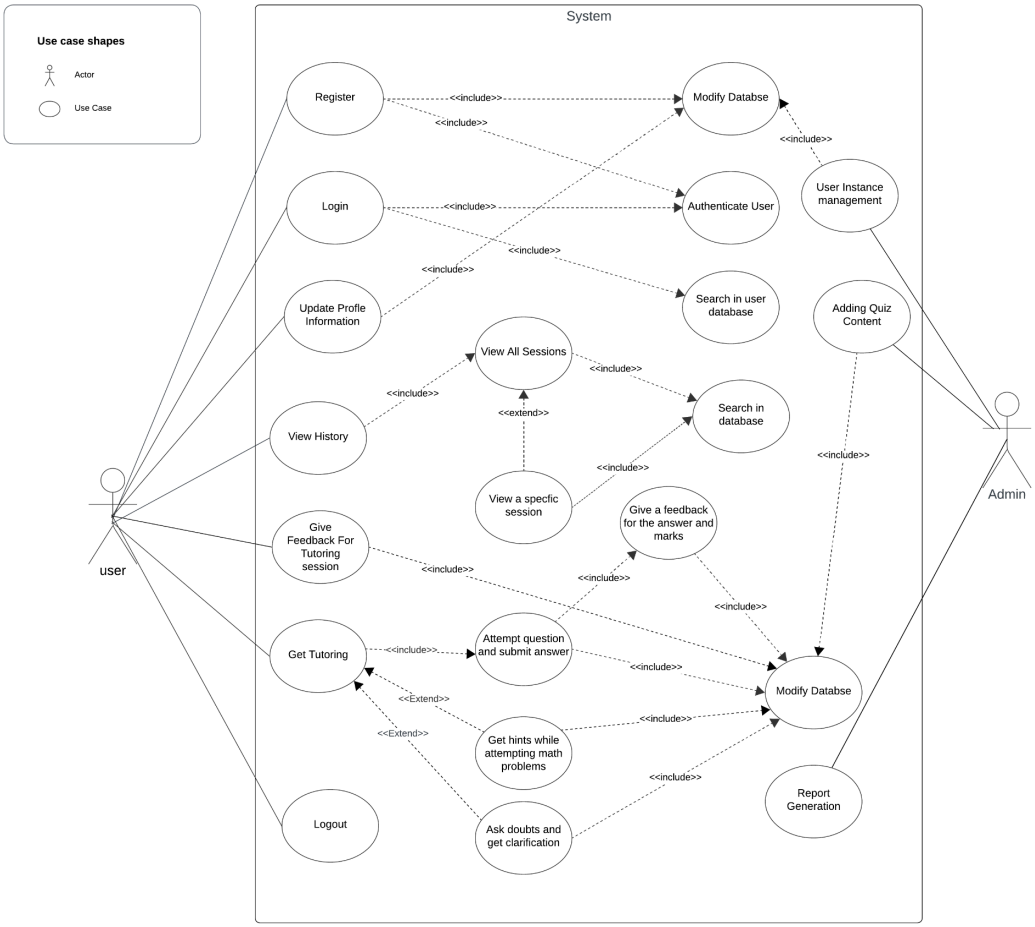

Figure 1: Use case Diagram

4.1. Use-Case Realizations

4.1.1. User Registration

Use case name	User Registration
Actor	User
Description	Joins new users to the system
preconditions	User must have an email
Main flow	User provide the email address System send a link to set password to the given email if there are no any user instance associated with the given email in the database

	User click on the given link in the email User provides password and confirmation password for the account System create a new user instance with the data System login the user
Successful end/post condition	User will be logged in
Fail end/post condition	User will receive error messages
Extensions	N/A

4.1.2. Login

Use case name	User Login and Password recovery
Actor	User
Description	To login to the system and recover the forgotten passwords.
preconditions	User should have already registered to the system
Main flow	User either provide the email and the password or go to the password recovery mode System finds the user instance associated with the email address. If the password is given, the system checks if the saved password is the same as the given password. If yes, the system logs in the user. If it is in the password recovery mode, the system sends an email containing a link to set a new password. User click on the link and set new password Then the user is logged in by the system.
Successful end/post condition	User will be logged in
Fail end/post condition	User will receive error message
Extensions	N/A

4.1.3. Selecting math problems and getting hints

Use case name	Selecting math problems and getting hints
Actor	User
Description	Provide user with the selected problem and hints to answer it
preconditions	User should have logged in to the system
Main flow	User select a problem by specifying a paper year and question number System takes the corresponding image and creates a new tutoring session object. System shows the user the question. If a user asks for hints, get the similar content in the VDB along with the answers, and send it to the LLM with proper prompt. Get the LLM response and present it to the user as tips while saving the response in the tutoring session object.
Successful end/post condition	User will get the math problem and hints
Fail end/post condition	User will receive error message
Extensions	N/A

4.1.4. Getting feedbacks on the answer

Use case name	Getting feedbacks on the answer
Actor	User
Description	Provide users with feedback, when users submit an answer for the math problem.
preconditions	User should have been logged in and user should have submit an answer for the problem
Main flow	System gets the answer from the user (image, canvas or text). If the answer is in image or canvas form, detect the text using OCR. From the VDB retrieve chunks which are similar to the text. System retrieves previous data stored in the tutoring session and feeds them along with the user's answer image and answer text and retrieves chunks to the LLM with proper prompt.

	Show the LLM feedback and marks to the user while saving these data in the tutoring session object.
Successful end/post condition	User will receive an feedback on answer
Fail end/post condition	Error
Extensions	N/A

4.1.5. Answering User Questions

Use case name	Answering User Questions
Actor	User
Description	System answers the questions raised by the user to clarify any issues.
preconditions	User should have logged in to the system and should have provided the answer to the current tutoring session
Main flow	System get user's question System get the previous data using the tutoring session System do a similarity search on VDB using the user's question and retrieve similar chunks Feed the user questions, retrieved chunks, and previous data to the LLM. Show the LLM response to the user while saving the conversation in the tutoring session. Repeat until the user stops asking questions.
Successful end/post condition	Users gets answers for their questions
Fail end/post condition	Error
Extensions	N/A

4.1.6. User Feedbacks

Use case name	User Feedbacks
Actor	User
Description	User is able to provide feedback on the expreience.

preconditions	User should have logged in to the system and user should have started a tutoring session and user should have got feedback on the answer
Main flow	Users will give feedback on the tutoring session (1 to 10). Current feedback will be presented to the user. System will save that data in the tutoring session object.
Successful end/post condition	User see the updated feedback
Fail end/post condition	Error
Extensions	N/A

4.1.7. Session History

Use case name	Session history
Actor	User
Description	Users are able to view their previous tutoring sessions.
preconditions	User should have logged in to the system
Main flow	Get all the tutoring sessions associated with the current user Present them all as a list and let the user choose one of them. View all data stored under that tutoring session to the user
Successful end/post condition	User will be able to see his/her past interactions
Fail end/post condition	Error
Extensions	N/A

4.1.8. Update Personal data

Use case name	Update personal data
Actor	User
Description	Allow user to update his/her personal information that was saved in the platform
preconditions	User should have logged in to the system

Main flow	System gets the corresponding user instance for the current user and show them to the user User update the information accordingly System update the user instance with the provided data.
Successful end/post condition	User information will be updated
Fail end/post condition	Error
Extensions	N/A

4.1.9. Using Math solving tool

Use case name	Using math solving tool
Actor	User
Description	Allow users to give their math problem to the system to solve it.
preconditions	User should have logged in to the system and user should have provide a math problem to the tool
Main flow	User provide the math problem System send the text to the math solving API System get the answer and show it to the user
Successful end/post condition	User gets the answers for the question
Fail end/post condition	Error
Extensions	N/A

4.1.10. Adding Quiz Content

Use case name	Adding Quiz Content
Actor	Admin
Description	Let the admin add new quiz content to the system
preconditions	Admin should have logged in to the system
Main flow	Admin first specify the year and the question number of the math problem. Admin provides the question image and the corresponding marking scheme image to the system. System stores the question in DB.

	System feeds the images to OCR models and gets text results. System presents them to the admin for approval. If the admin approves the results, the system will add the text to the VDB.
Successful end/post condition	New content will be added to the system
Fail end/post condition	Error
Extensions	N/A

4.1.11. User Instance Management

Use case name	User Instance Management
Actor	Admin
Description	Allow the admin to remove user instances
preconditions	Admin should have logged in to the system
Main flow	System gets all the user instances and present it to admin as a list Admin select the user instances admin wants. Admin select delete option. System delete the selected user instances
Successful end/post condition	User instances will be deleted
Fail end/post condition	User instances will not be deleted and error message will be shown
Extensions	N/A

4.1.12. Report Generation

Use case name	Report Generation
Actor	Admin
Description	Let admin generate reports about system
preconditions	Admin should have logged in to the system
Main flow	System should take all LLM query and responses. System should send it to the langfuse platform. Admin will generate reports in langfuse platform
Successful end/post condition	Admin generates reports

Fail end/post condition	Error State
Extensions	N/A

5. Logical View

5.1. Overview

EduGenius will be decomposed into the following logical layers:

Presentation Layer: Handles user interface elements, user interaction, and data visualization.

Application Logic Layer: Implements core functionalities like problem retrieval, answer evaluation, feedback generation, and interaction with the LLM and data storage.

Data Management Layer: Encapsulates database interactions for user management, problem storage, and retrieval.

Integration Layer: This layer handles the interaction between system and external dependencies such as LLMs and vision models.

These layers are further discussed from the implementation point of view under section 8 - Implementation view.

5.2. Architecturally Significant Design Packages

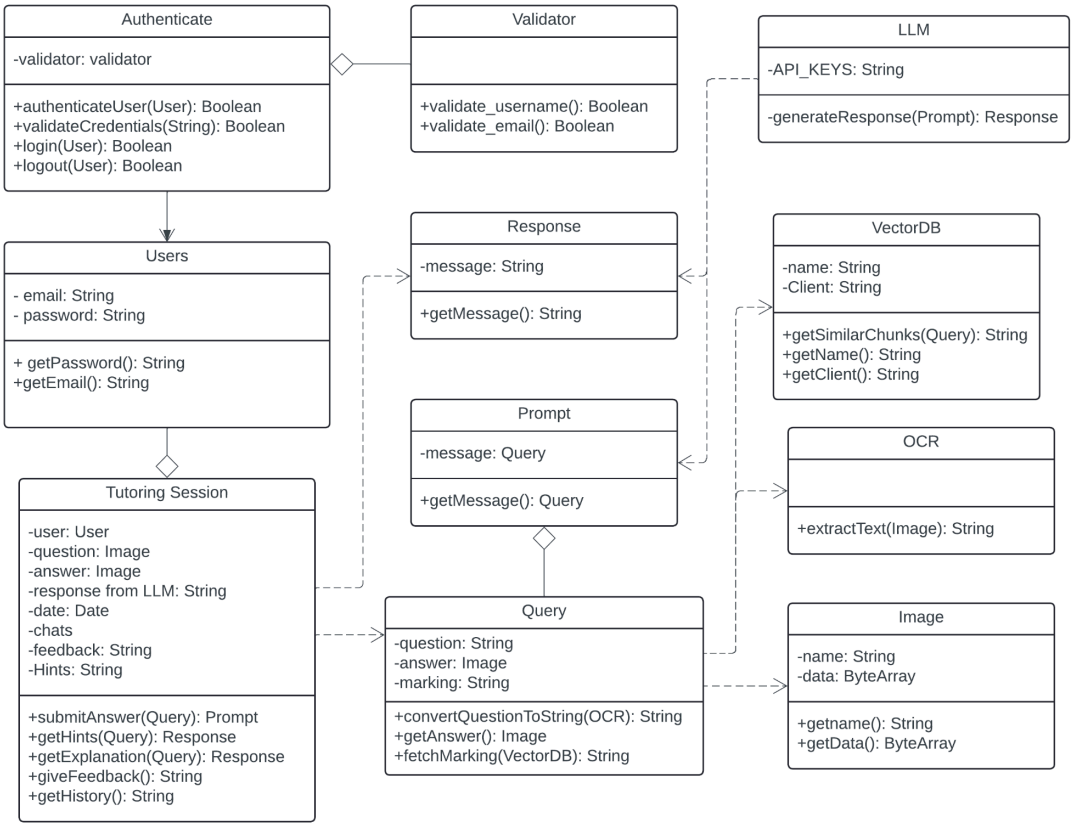

Figure 2: Logical View

- **Application Logic Layer**
  - **Validate:** This class is responsible for checking validity of user name and user email.
  - **Authenticate:** This handles the user login(validate credentials), logout and registration. It uses the validator class for validating user email address and user name during registration.
  - **User:** represents the user entity. This stores the email address and password of the user in its fields.
  - **Tutoring Session:** This class represents each tutoring session. It stores the prompts and responses generated by the user and LLM during tutoring interaction. All the functionality provided under tutoring will be implemented under this class.
  - **Prompt:** This represents the prompt generated by the system on behalf of the user. This contains a question and answer(as String) and is submitted to the LLM class.

    - **Query:** This class contains the image of the question that user is attempting and the answer image generated by the user instance during the tutorial session.
    - **Response:** This class represents the response generated by LLM.
    - **Image:** Image class represents the image data in the system. Both the user's answer and question images.

- **Integration Layer**
    - **OCR:** This OCR class is to represent the OCR component in the system. It is used to manage the interaction between system and vision models. This OCR component is responsible for extracting text content from the image.
    - **LLM:** The LLM class represents the LLM. It manages the interaction between the interaction between the system and external LLMs.

- **Data Management Layer**
    - **VectorDB:** This is to represent and manage the vector database instance.

6.Process View

6.1. User Registration Process

A. Activity Diagram

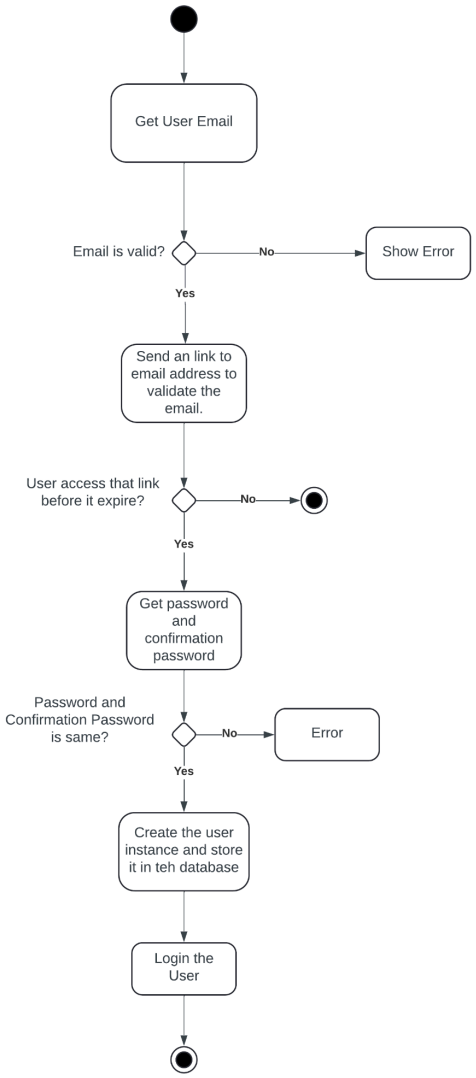

Figure 3: Activity diagram of user registration process

B. Sequential Diagram

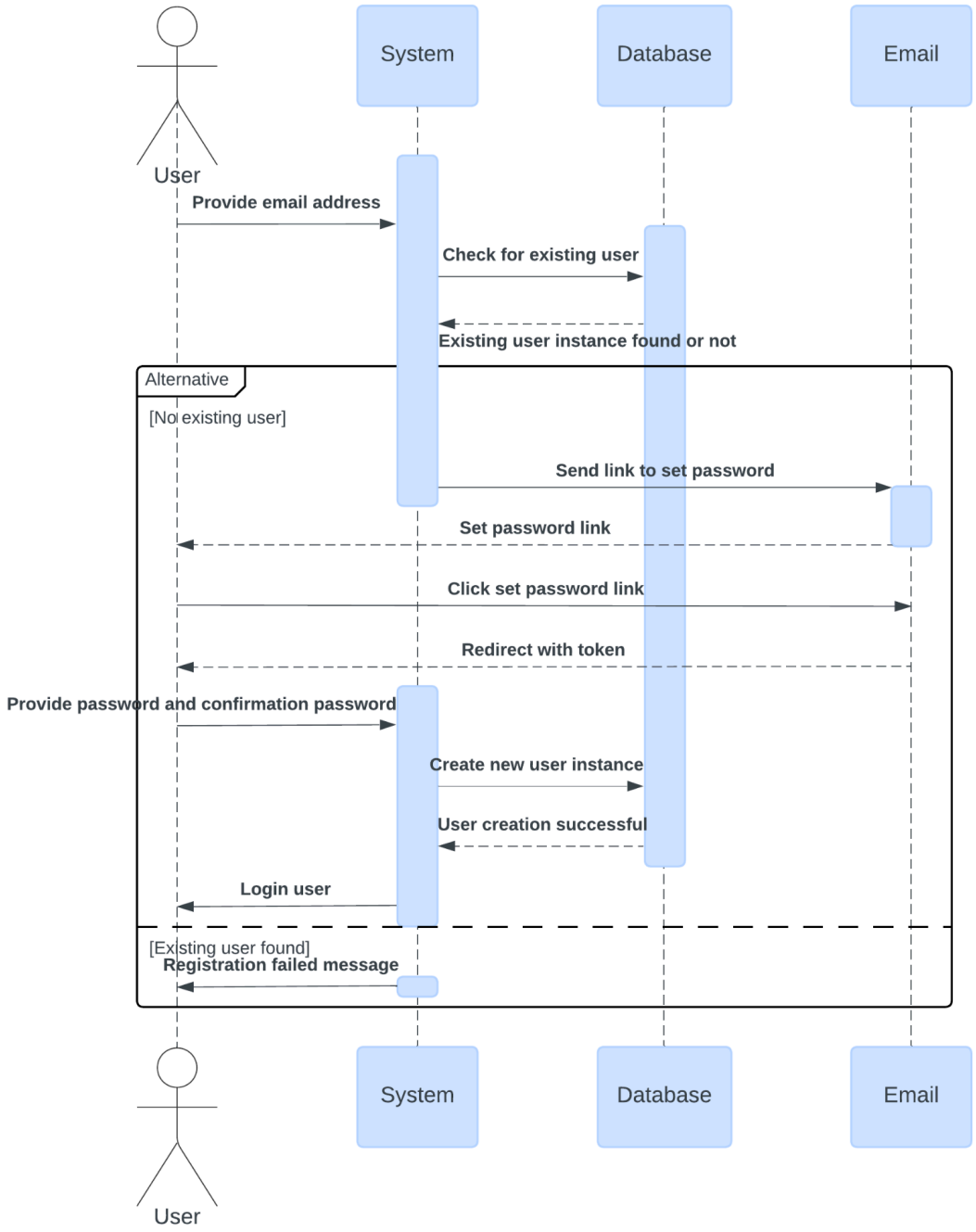

Figure 4: Sequential diagram of user registration process

6.2. User Login and Password Recovery Process

A. Activity Diagram

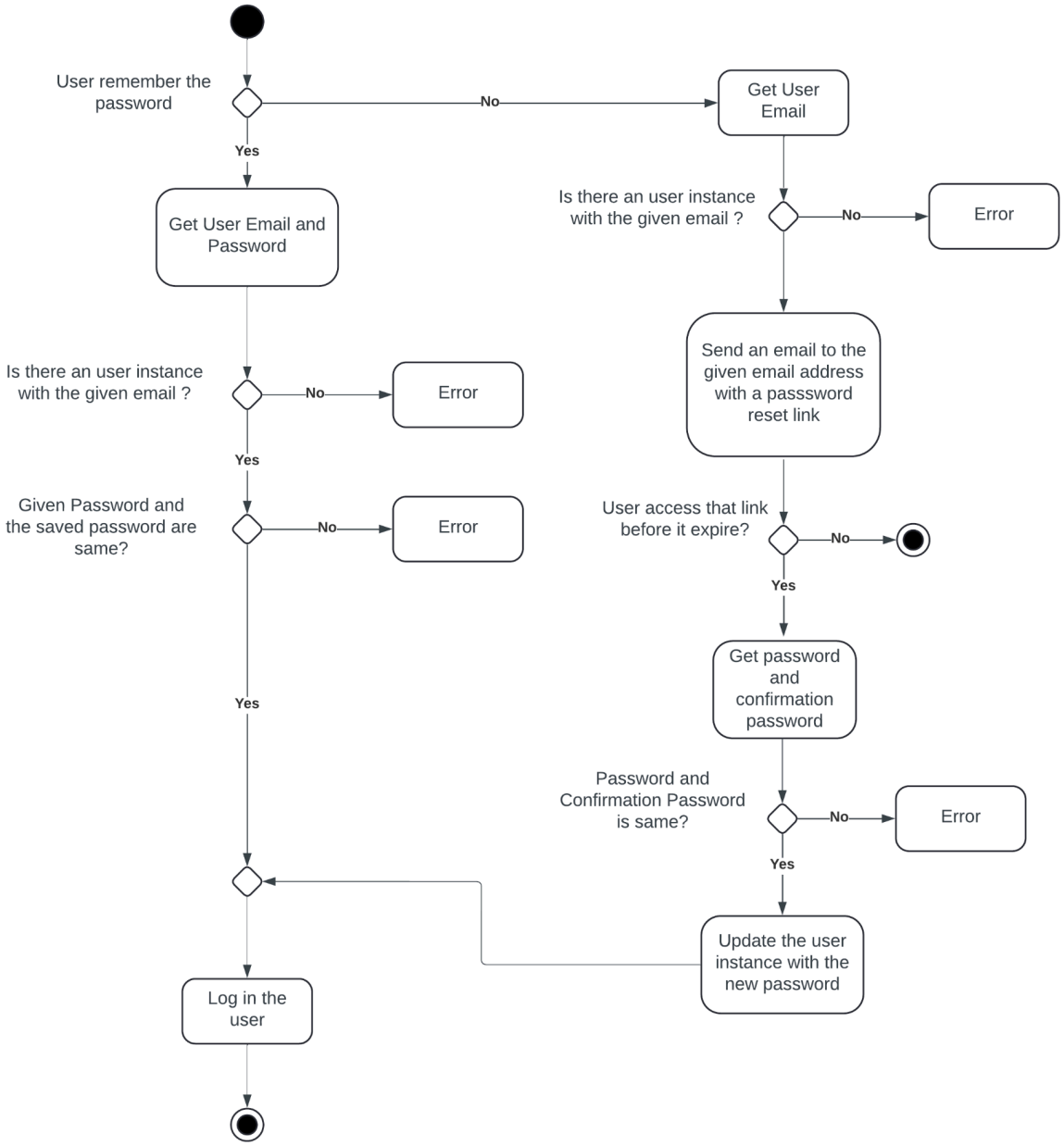

Figure 5: Activity diagram of user login and password recovery process

B. Sequential Diagram

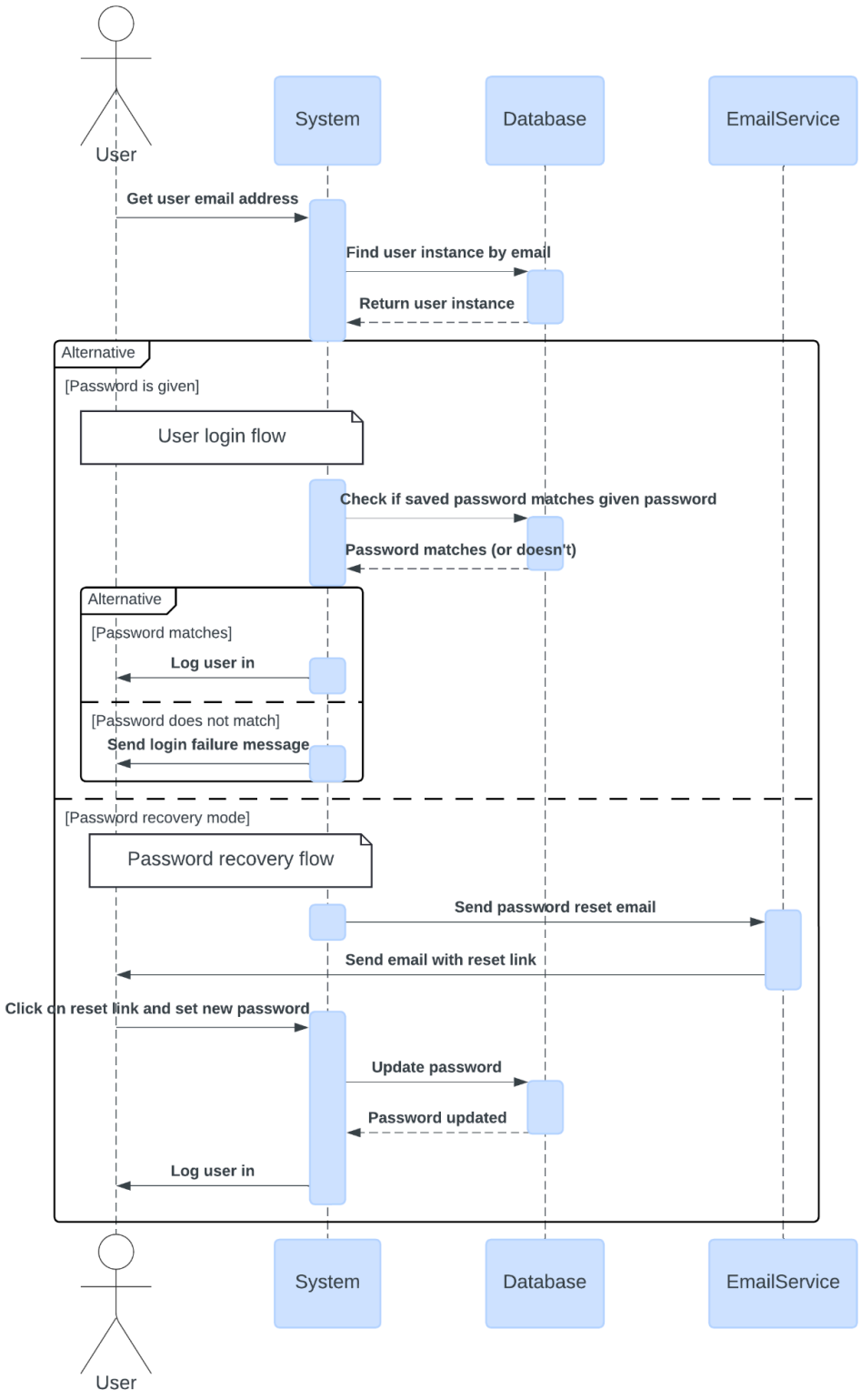

Figure 6: Sequential diagram of user login and password recovery process

6.3. Problem Selecting and Hint Generation Process

A. Activity Diagram

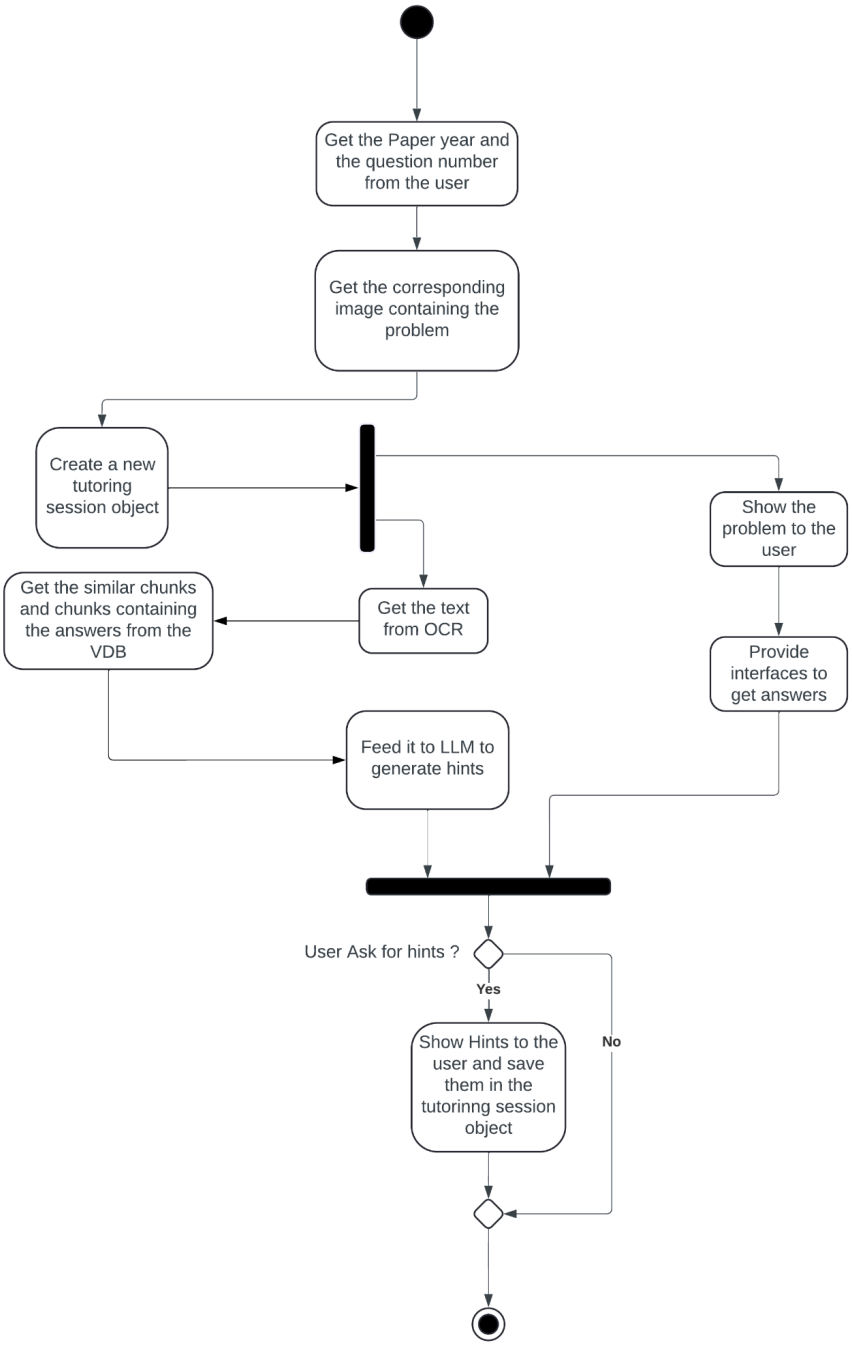

Figure 7: Activity diagram of problem selecting and hint generation process

B. Sequential Diagram

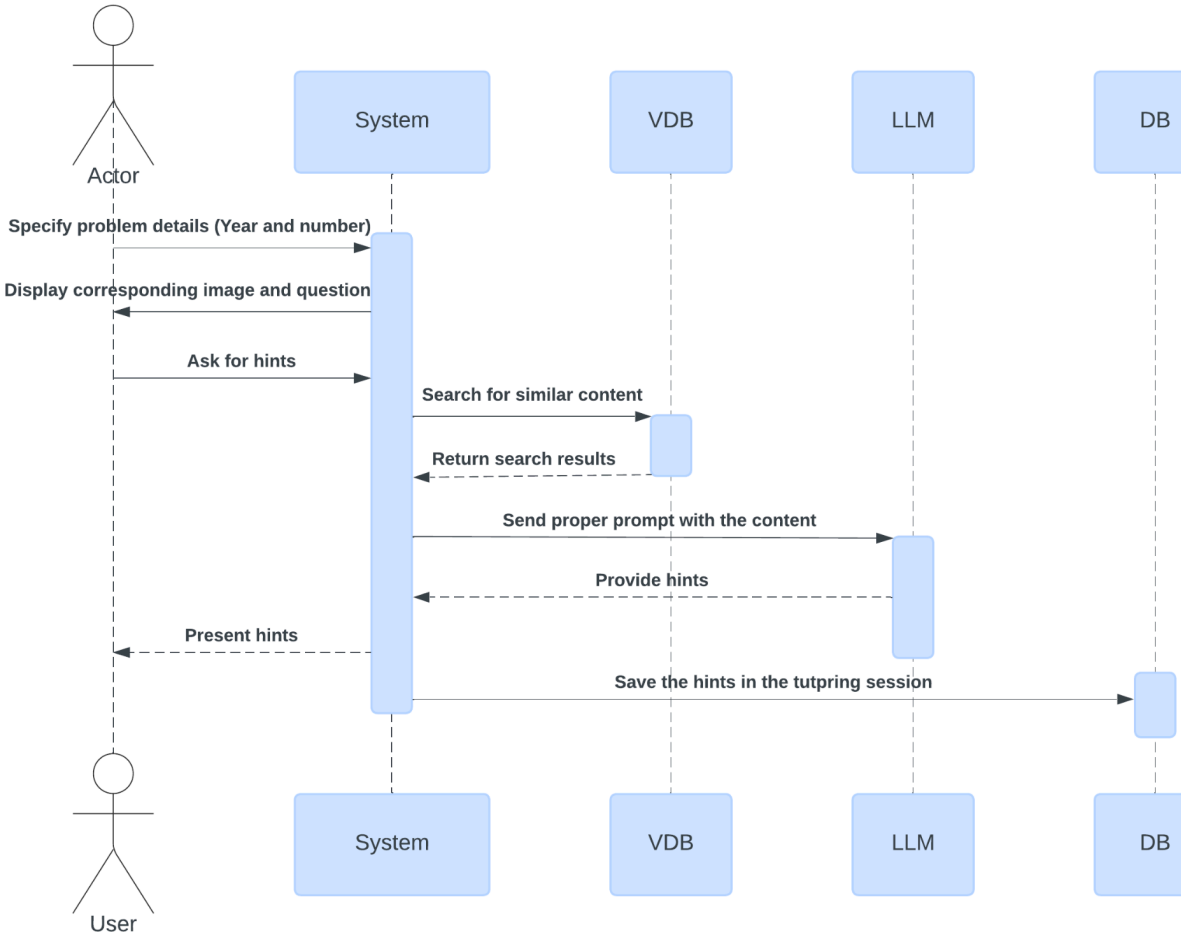

Figure 8: Sequential diagram of problem selecting and hint generation process

6.4. Feedback Generation and Marking Process for User Answer

A. Activity Diagram

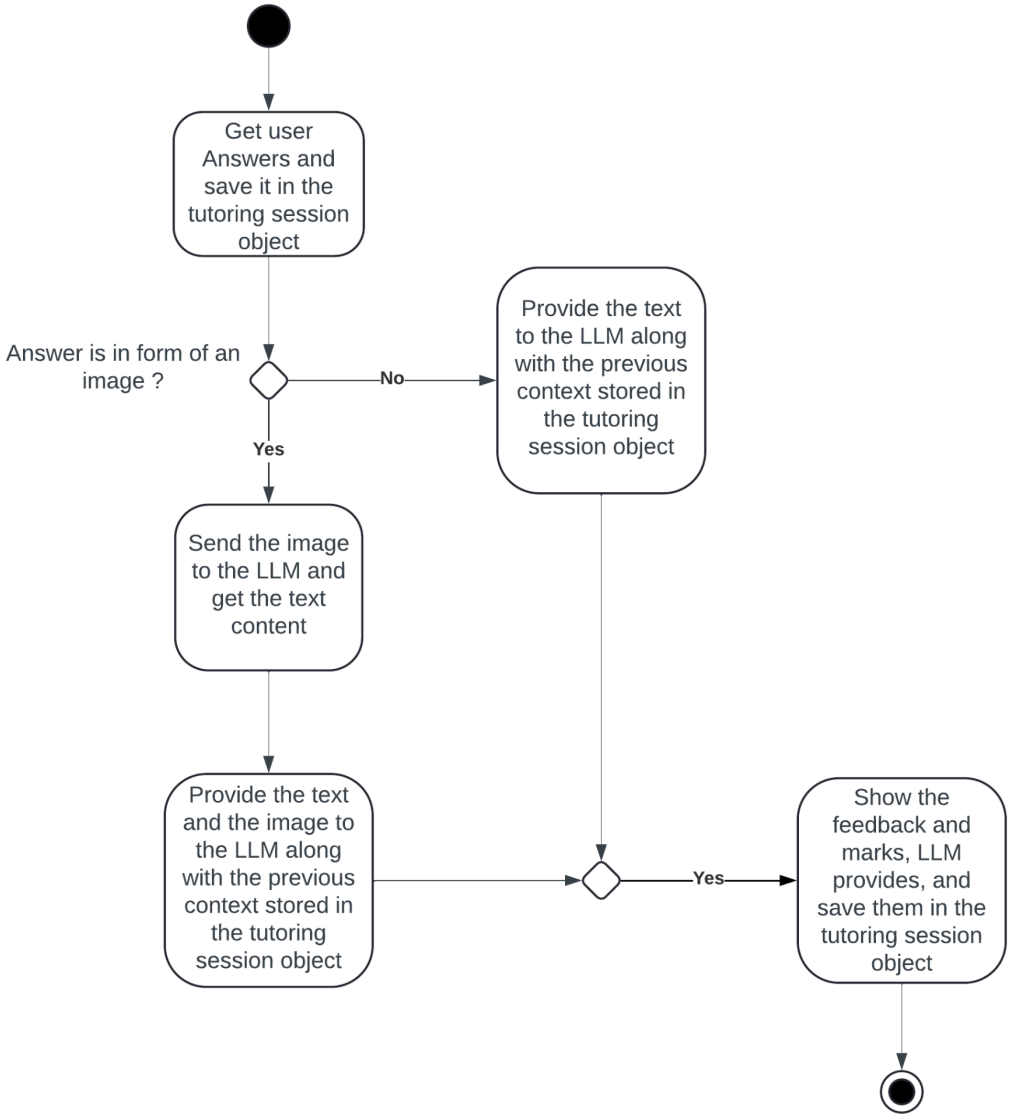

Figure 9 : Activity diagram of feedback generation and marking process for user answer

B. Sequential Diagram

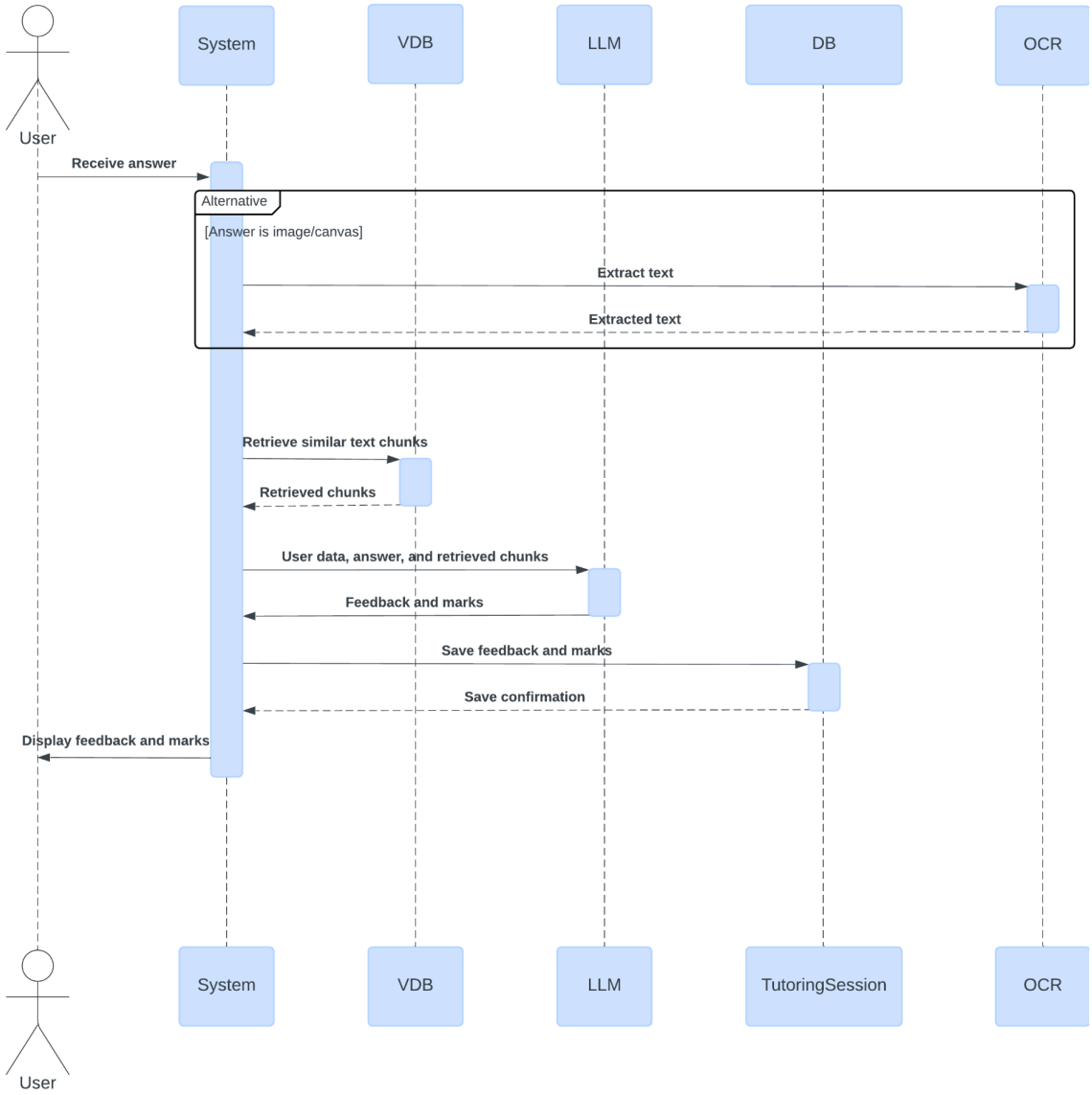

Figure 10: Sequential diagram of feedback generation and marking process for user answer

6.5. Answer Generation Process for User Questions

A. Activity Diagram

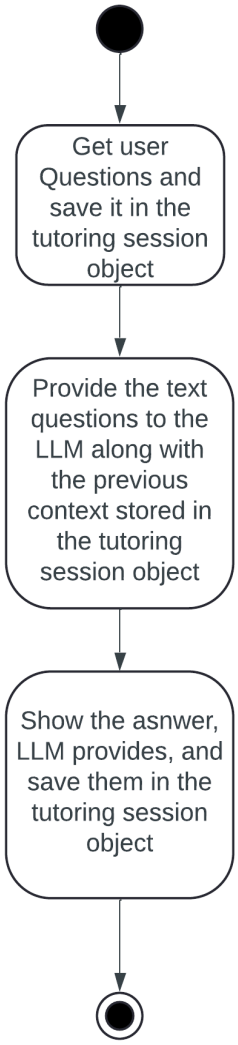

Figure 11: Activity diagram for answer generation process for user questions

B. Sequential Diagram

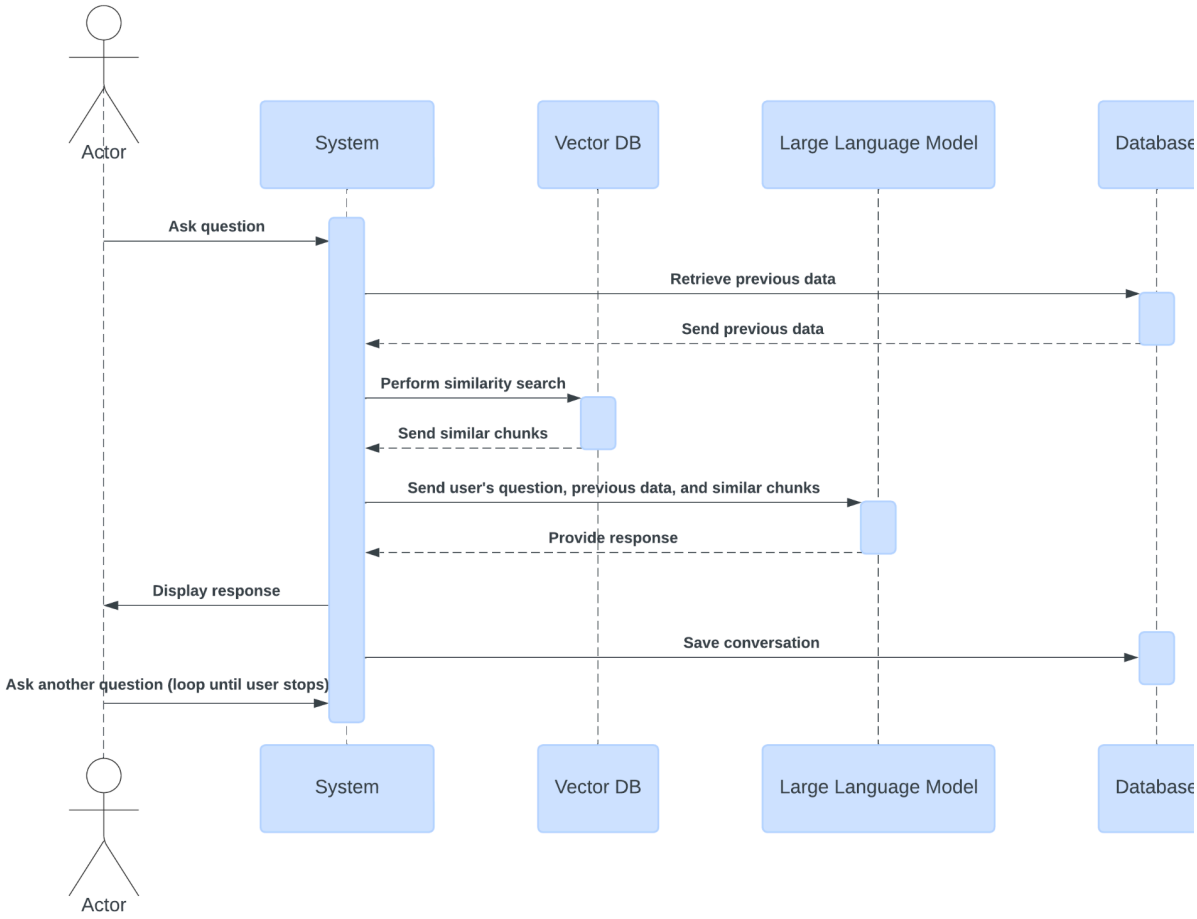

Figure 12: Sequential diagram for answer generation process for user questions

6.6.User Review Process

A. Activity Diagram

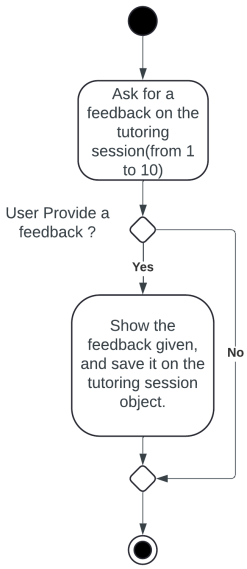

Figure 13: Activity diagram for user review process

B. Sequential Diagram

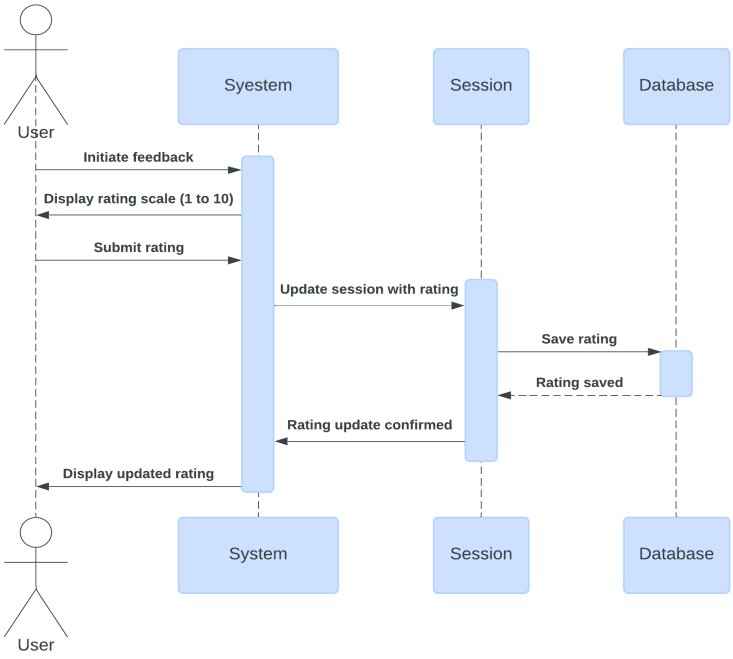

Figure 14: Sequential diagram for user review process

6.7.User History Management Process

A. Activity Diagram

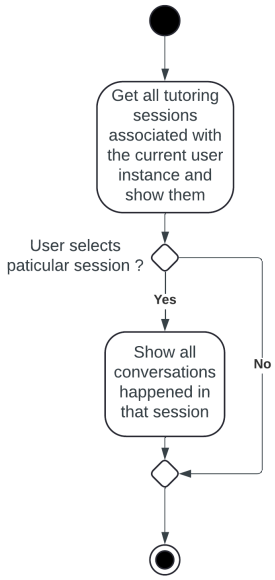

Figure 15: Activity diagram for user history management process

B. Sequential Diagram

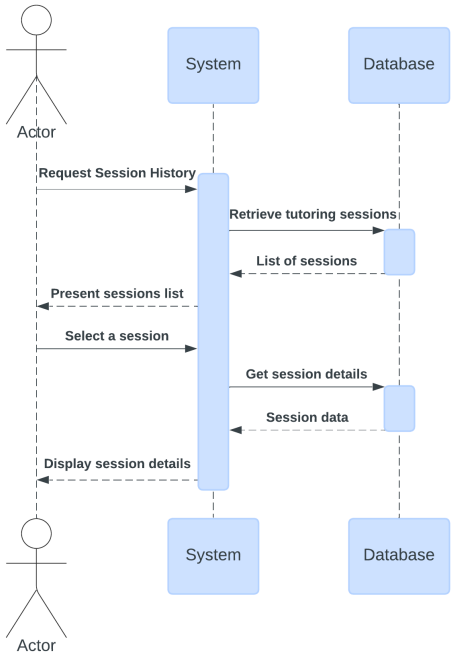

Figure 16: Sequential diagram for user history management process

6.8. User Information Updating Process

A. Activity Diagram

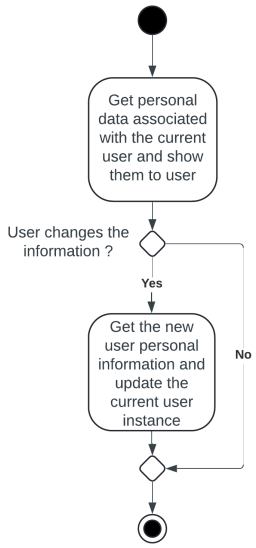

Figure 17: Activity diagram for user information updating process

B. Sequential Diagram

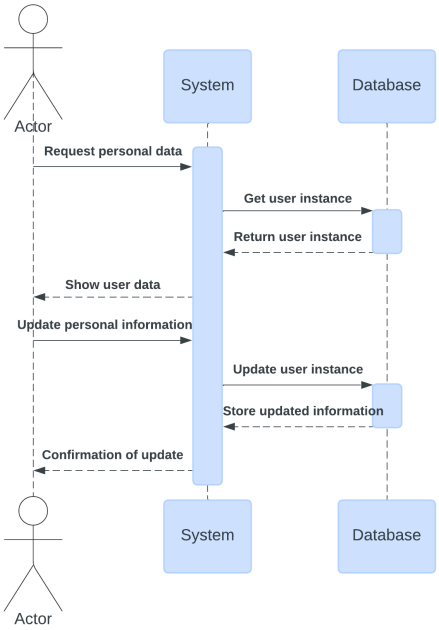

Figure 18: Sequential diagram for user information updating process

6.9. Math Problem Solving Process

A. Activity Diagram

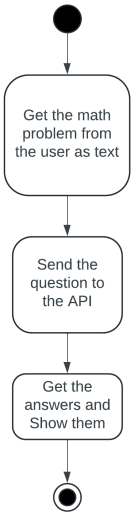

Figure 19: Activity diagram for math problem solving process

B. Sequential Diagram

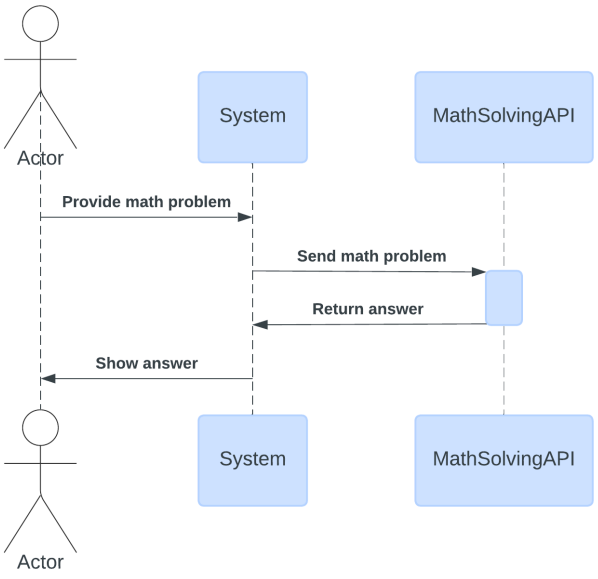

Figure 20: Sequential diagram for math problem solving process

6.10.Quiz Content Adding Process

A. Activity Diagram

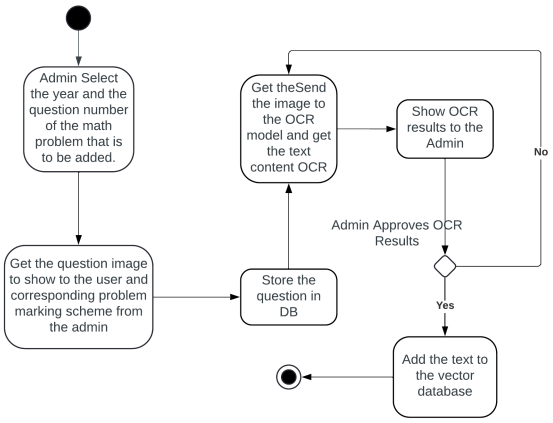

Figure 21: *Activity diagram for quiz content adding process*

B. Sequential Diagram

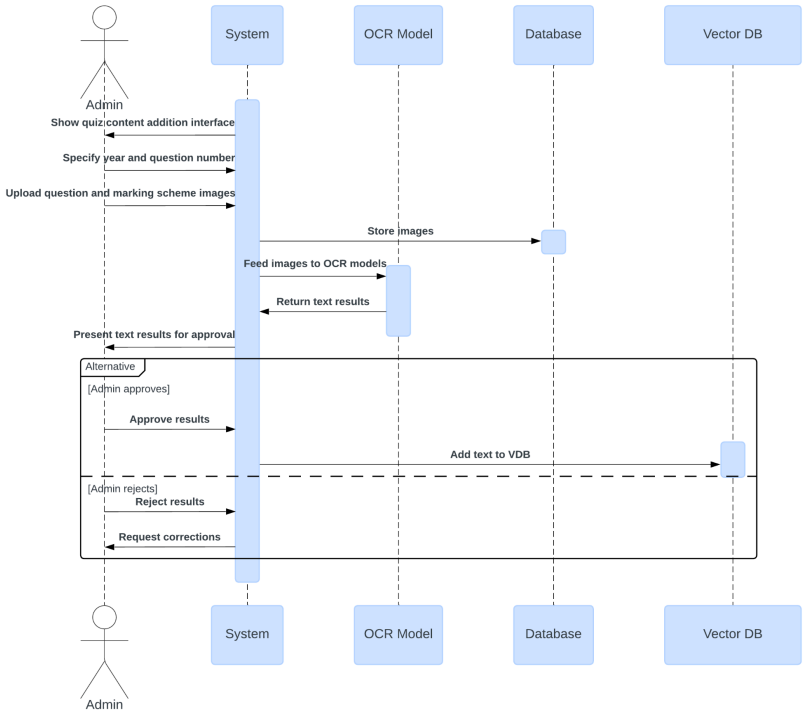

Figure 22: Sequential diagram for quiz content adding process

6.11. User Instances Management Process

A. Activity Diagram

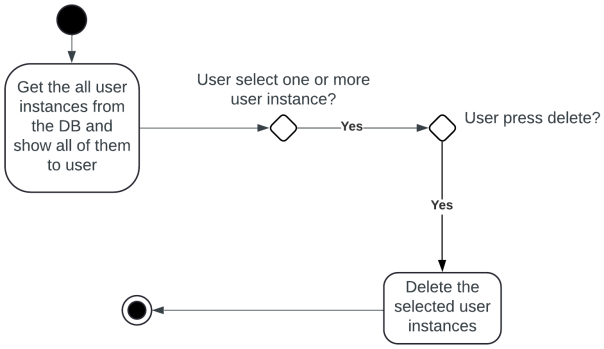

Figure 23: Activity diagram for user instances management process

B. Sequential Diagram

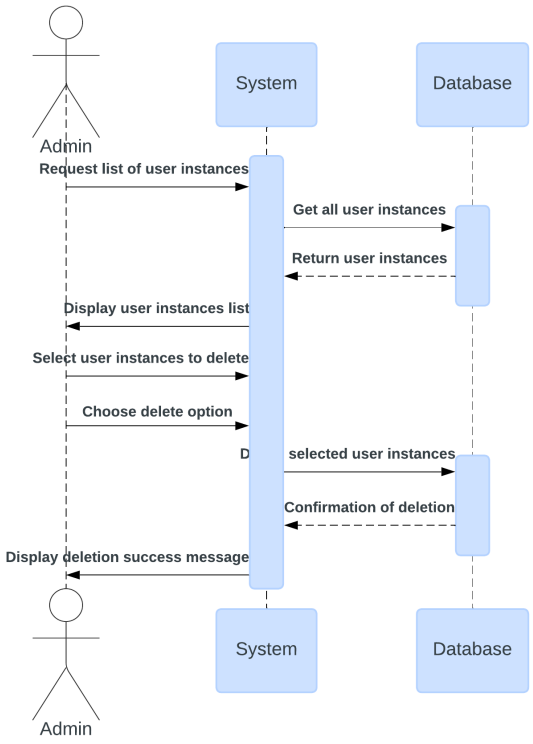

Figure 24: Sequential diagram for user instances management process

6.12. Report Generating Process

A. Activity Diagram

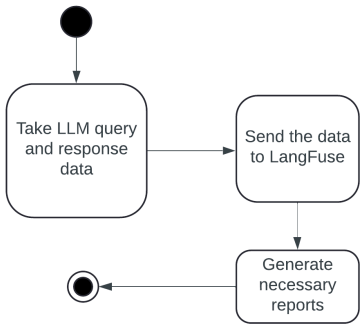

Figure 25: Activity diagram for report generation process

B. Sequential Diagram

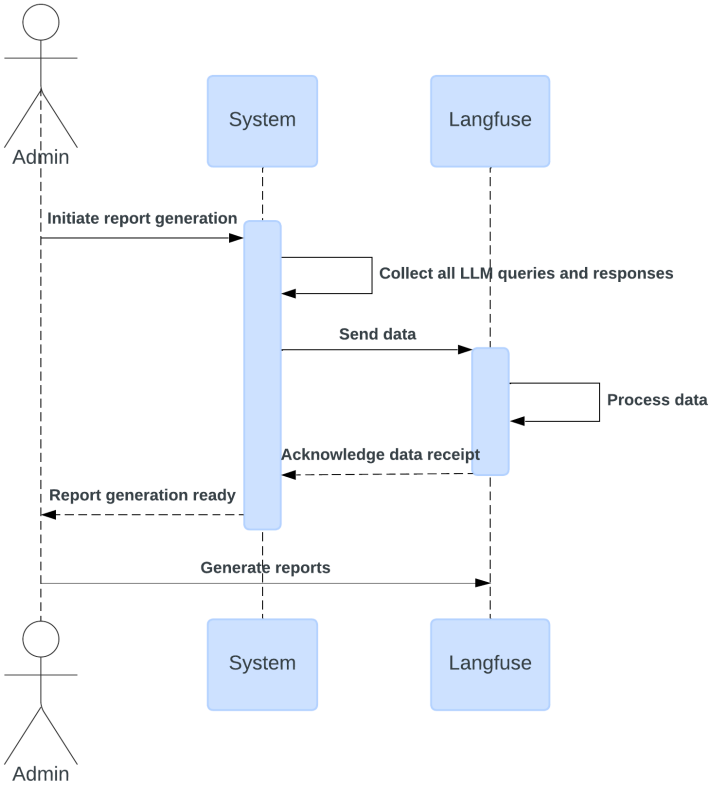

Figure 26: Sequential diagram for report generation process

7. Deployment View

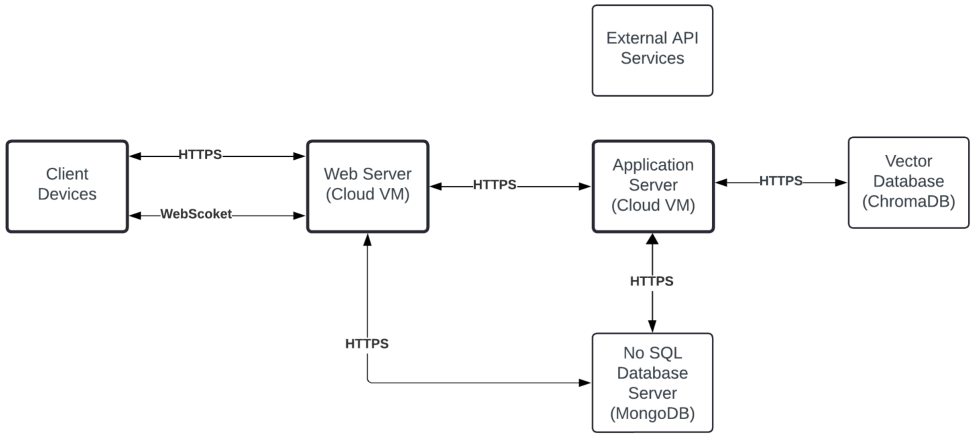

Figure 27: Deployment Diagram

- **Client Devices:** Access the Edugenius web application via web browsers.
- **Web Server:** Serves web content (HTML, CSS, JavaScript) and application logic to the user's browser.
- **Application Server:** Processes application logic, interacts with the database, and generates dynamic content. (Python with Streamlit)
- **Database:** Stores application data (MongoDB)
- **Vector Database:** Stores the textual data as embeddings.
- **External APIs:** Provide LLM access, Vision Model access and other additional services.

8. Implementation View

8.1. Overview

This section outlines the overall structure of the implementation model, detailing the decomposition of the software into layers and subsystems, and describing the architecturally significant components ensuring modularity, scalability, and maintainability.

8.2. Layers

EduGenius will be organized into the following logical layers:

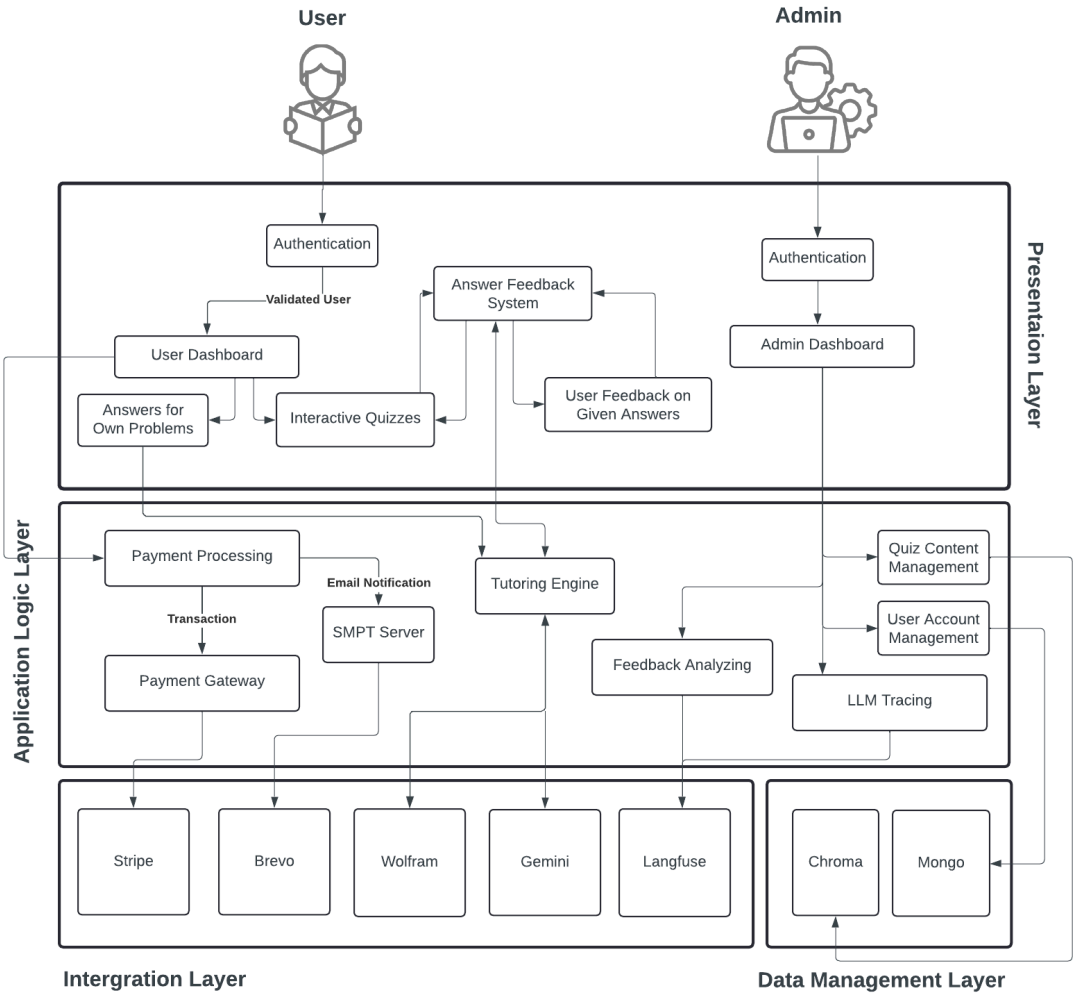

Figure 28: Implementation view

1. **Presentation Layer:** Manages user interface elements, user interactions, and data visualization.
    - **User Dashboard/ Admin Dashboard:** Provides a user-friendly interface for users to access the system's functionalities.
    - **Interactive Quizzes:** Provides an interface for users to take quizzes and receive feedback on their performance interactively.
    - **Answers for Own Problems:** Provides a chatbot interface for users to ask questions which are raised by themselves while solving problems.

2. **Application Logic Layer:** Implements core functionalities such as handling tutoring engine, feedback analyzing and payment processing.
    - **Tutoring Engine:** Provides the core functionality of the system, including problem-solving, feedback generation, and user interaction.
    - **Feedback Analyzing:** Analyzes user feedback to improve the tutoring engine and user experience.
    - **Payment Processing:** Manages user payments for premium features and services.
    - **Quiz Content Management:** Manages quiz content, including question generation, quiz generation, and quiz grading.
    - **User Account Management:** Manages user accounts, basically user banning, user account deletion, and restricting user access.
    - **LLM Output Tracing:** Traces the output of the LLMs to improve the tutoring engine and user experience.

3. **Data Management Layer:** Handles database interactions for user management, and problem storage
    - **MongoDB:** Stores user data, progress and history
    - **ChromaDB:** Stores problem data, including problem descriptions, and marking schemes.

4. **Integration Layer:** Manages interactions between the system and external dependencies, including LLMs and vision models.
    - **Gemini:** Manages interactions with the LLMs for problem-solving and feedback generation
    - **Wolfram:** Manages interactions with the Wolfram API for problem-solving and feedback generation
    - **Lanfuse:** Gathers user feedback and traces the output of the LLMs to improve the tutoring engine and user experience.
    - **Brevo:** SMTP server for sending emails to users.
    - **Stripe:** Manages payment processing for premium features and services.

9. Size and Performance

The EduGenius Mathematics Tutoring Application is designed to handle substantial data and user interactions. The major dimensioning characteristics that impact the architecture include:

1. **User Capacity**: The system is designed to support up to 1,000 concurrent users.
2. **Database Size**: Expected to store millions of records for user profiles, mathematical problems, tutoring sessions, and feedback.
3. **Response Time**: Critical operations, such as problem retrieval, answer evaluation, and feedback generation, are optimized to ensure a response time of under 3 minutes.
4. **Throughput**: The system should efficiently handle multiple simultaneous interactions with external APIs, such as LLMs and math solving tools, without significant performance degradation.
5. **Scalability**: The microservices architecture, usage of NOSQL databases, and containerization with Docker allow for horizontal scaling to accommodate increased load by adding more instances of the required services.

10. Quality

The software architecture of EduGenius is designed to ensure several key quality attributes:

1. **Extensibility**: The modular design allows for easy addition of new features and functionalities without significant changes to the existing system.
2. **Reliability**: The architecture includes redundant components and failover mechanisms to achieve a Mean Time Between Failures (MTBF) of at least 3,000 hours and a Mean Time To Repair (MTTR) of less than 7 hours.
3. **Portability**: The use of standard technologies and containerization with Docker ensures that the application can be easily deployed across different environments.
4. **Security and Privacy**: Compliance with GDPR, COPPA, and FERPA ensures robust data protection and user privacy. The system employs HTTPS for secure communication and includes comprehensive user identity verification measures.
5. **Maintainability**: Adherence to coding standards, thorough documentation, and the use of version control (Git) facilitate easy maintenance and updates.
6. **Scalability**: The microservices architecture and cloud deployment capabilities allow the system to scale horizontally to meet increasing demand without sacrificing performance.
7. **Usability**: The consistent and intuitive user interface ensures that users can navigate the system with minimal training, enhancing the overall user experience.

11. References

[1] “Introduction | 🔗 LangChain.” https://python.langchain.com/v0.2/docs/introduction/ (accessed Jul. 07, 2024).

[2] “Chroma Docs.” https://docs.trychroma.com/ (accessed Jul. 07, 2024).

[3] “What is MongoDB? - MongoDB Manual v7.0.” https://www.mongodb.com/docs/manual/ (accessed Jul. 08, 2024).

[4] “Gemini API Developer Docs and API reference,” Google for Developers. https://ai.google.dev/gemini-api/docs (accessed Jul. 08, 2024).

[5] “Get started with LangSmith | 🦜🛠️ LangSmith.” https://docs.smith.langchain.com/ (accessed Jul. 08, 2024).

[6] “Langfuse documentation - langfuse.” https://langfuse.com/docs (accessed Jul. 08, 2024).

[7] “Get started with Streamlit - Streamlit Docs.” https://docs.streamlit.io/get-started (accessed Jul. 08, 2024).

[8] “Wolfram|Alpha Show Steps API: Reference & Documentation.” https://products.wolframalpha.com/show-steps-api/documentation (accessed Jul. 09, 2024).

[9] Modran, H., Bogdan, I. C., Ursuțiu, D., Samoila, C., and Modran, P. L., "LLM Intelligent Agent Tutoring in Higher Education Courses using a RAG Approach," Preprints, 2024. [Online]. Available: https://doi.org/10.20944/preprints202407.0519.v1 (accessed Jul. 10, 2024).

[10] “Build a Q&A App with Multi-Modal RAG using Gemini Pro | Google Codelabs,” Google Codelabs. https://codelabs.developers.google.com/multimodal-rag-gemini#4 (accessed Jul. 10, 2024).