FP25-119-D-Rehnuma

Project Team

Faseeh Iqbal 22I-1856
Ahmad Hasan 22I-1945
Manhab Zafar 22I-1957

Session 2022-2026

Supervised by

Ms. Amna Irum

Co-Supervised by

Dr. Qurut-ul-Ain

Department of Data Science And Artificial Intelligence

National University of Computer and Emerging Sciences
Islamabad, Pakistan

June, 2026

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this project proposal is to present "Rehnuma", an intelligent animated tutoring system designed to assist students in understanding advanced statistical concepts with greater clarity and efficiency. In the current educational environment, students often face significant challenges when dealing with probability and statistics, particularly during their initial semesters where prior exposure is limited. Traditional learning resources such as static notes, lengthy textbooks, or unstructured online videos often fail to provide personalized and interactive explanations. Rehnuma bridges this gap by offering a system that dynamically generates animated videos, coupled with explanatory narration, tailored to specific queries raised by students. By integrating large language models (LLMs), retrieval-augmented generation (RAG) pipelines, and Manim animations, the system ensures that learners receive accurate, visual, and concept-based explanations. This approach not only makes abstract concepts easier to grasp but also enhances long-term retention. The proposed system will ultimately create a self-adaptive, accessible, and interactive platform that aligns with modern digital learning trends. **?**.

1.1 Existing Solutions

Several online learning platforms currently provide mathematics and science education, but they present notable limitations in personalization and real-time interaction. The two leading solutions are:

- **Khan Academy**
  Offers a large library of pre-recorded video lessons with clear narration, structured topic progression, practice exercises, and limited quiz integration.

- **Brilliant.org**
  Provides pre-designed interactive lessons, rich quiz integration, problem-solving-focused courses, and supporting text/narration. Personalization is restricted to predefined course paths.

Table 1.1: Comparison of Existing Solutions

System Name	System Overview	System Limitations
Khan Academy	• Pre-recorded video explanations with narration • Structured courses and practice exercises • Limited quiz integration	• No personalization • No support for user-input queries • No real-time Q&A • Pre-recorded content only
Brilliant.org	• Pre-designed interactive explanations • Strong quiz integration • Structured, problem-solving-focused courses • Narration/supporting text	• Limited personalization (only within predefined courses) • No support for arbitrary user-input queries • No real-time Q&A

1.2 Problem Statement

Students entering advanced courses in probability and statistics frequently encounter difficulties due to their limited background in conceptual mathematics. These difficulties often lead to surface-level understanding and a lack of problem-solving confidence. While numerous online resources such as YouTube lectures and digital textbooks exist, they require students to spend a considerable amount of time searching for the right content, which is often not aligned with their exact query. Moreover, most of these resources present content in a generic, onesize-fits-all manner that does not adapt to a learner's specific knowledge gaps. This inefficiency reduces the effectiveness of independent learning, especially for students who require instant clarity. Students may also face cognitive overload while navigating long lectures for a single concept. The absence of interactive and context-aware tutoring tools contributes to weak conceptual foundations, resulting in poor

academic performance. Rehnuma addresses this problem by providing an on-demand intelligent tutor capable of delivering personalized animated explanations directly based on student queries. By doing so, the system eliminates the inefficiencies of manual content search and ensures that students receive focused, visual, and clear explanations.

1.3 Scope

The scope of Rehnuma encompasses the development of a fully functional prototype that demonstrates the generation of animated explanations for queries. The main focus lies in probability and statistics concepts, ensuring that the content generated is accurate, relevant, and comprehensible. The system will include functionalities such as query-based animated video generation, quiz generation, real-time QA, personalized dashboard, and a classroom. Scope boundaries include limiting the dataset to curated problems from selected textbooks and restricting animation generation to Manim-compatible scripts. The solution will not attempt to replace human instruction but rather supplement it by providing students with an intelligent self-learning platform. This ensures a clear boundary between automated explanations and human-led discussions, making the system a supportive tool rather than a replacement. The system will demonstrate the feasibility of integrating LLMs, RAG pipelines, and Manim to create an adaptive educational tool that enhances independent learning, improves conceptual clarity, and promotes confidence in tackling mathematical problems.

1.4 Modules

1.4.1 Module 1: Dataset Pipeline

This module is responsible for creating and managing the core dataset required for query-based explanations. It ensures structured storage of problems, solutions, and animation codes in a retrievable format for efficient usage in the system.

1. Generate a dataset using DeepSeek Math from textbook solutions.
2. Store problems with multiple keys (description, code, etc.) in JSON format.
3. Implement ChromaDB for vector storage and retrieval

1.4.2 Module 2: Animation Pipeline

The animation pipeline generates customized visual explanations based on student queries. It integrates solution generation, reference retrieval, and code rendering to create synchronized video and audio explanations.

1. Use DeepSeek Math for solution generation.
2. Retrieve reference problems through RAG.
3. Employ DeepSeek Code for generating Manim-compatible code.
4. Render animations with synced audio explanations using gtts (Python library).

1.4.3 Module 3: Quiz System

This module reinforces learning by generating quizzes linked to student queries. It allows students to test their understanding and track progress through structured evaluations.

1. Generate quizzes from user queries using DeepSeek Math.
2. Provide multiple-choice questions (MCQs) with scoring.
3. Integrate with the dashboard for progress tracking.

1.4.4 Module 4: Real-Time QA System

The QA system enables instant responses to student doubts during or after lessons. It ensures continuity in the learning process through text-based, real-time interaction.

1. Use DeepSeek Math for real-time responses.
2. Provide answers in text format.

1.4.5 Module 5: Personalized Dashboard

This module offers personalized tracking of a student's progress and performance. It ensures learners receive feedback and insights into their strengths and weaknesses.

1. Display quiz history and performance analytics.
2. Maintain personalized progress records.
3. Provide visual feedback to learners.

1.4.6 Module 6: Classroom

The classroom module supports collaborative learning by enabling group-based participation. It allows users to share learning experiences and benefit from interactive discussions.

1. Develop a classroom for collaboration.
2. Users will be able to invite other users and watch animations together.

Chapter 2

Project Requirements

2.1 Use-case/Event Response Table/Storyboarding

Since Rehnuma is an interactive end-user application, the use case approach is most suitable. The system involves multiple user interactions across different modules, making use case diagrams and detailed use cases the ideal requirement gathering technique.

1. Generate Animated Explanation
2. Take Quiz
3. Ask Real-Time Question
4. View Progress Dashboard
5. Join Classroom Session
6. Create Dataset Entry (Admin)

2.2 Functional Requirements

2.2.1 Module 1: Dataset Pipeline

Following are the requirements for module 1:

FR1.1:Dataset Generation The system shall allow administrators to input textbook problems and solutions, which will be processed by DeepSeek Math to generate structured dataset entries containing problem descriptions, solutions, and metadata.

FR1.2 Code Generation and Validation The system shall use DeepSeek Math to generate Python code from problem-solution pairs and validate the generated code using SymPy and NumPy libraries before storage.

FR1.3 Manim Code Generation The system shall automatically generate Manim-compatible animation code and associated metadata for each validated problem-solution pair using Code-Llama.

FR1.4 JSON Storage The system shall store all dataset entries in JSON format with multiple searchable keys including problem description, solution steps, Manim code, difficulty level, topic tags, and unique identifiers.

FR1.5 Vector Database Integration The system shall implement ChromaDB for vector storage, enabling semantic search and efficient retrieval of similar problems based on query embeddings.

FR1.6 Dataset Versioning The system shall maintain version control for dataset entries, allowing administrators to update, modify, or deprecate entries while preserving historical data.

2.2.2 Module 2: Animation Pipeline

Following are the requirements for module 2:

FR2.1 Query Processing The system shall accept natural language queries from students and process them through prompt engineering to enhance retrieval accuracy.

FR2.2 RAG-based Retrieval The system shall retrieve the top-k most relevant reference problems from ChromaDB based on semantic similarity to the user's query.

FR2.3 Solution Generation The system shall generate step-by-step mathematical solutions using DeepSeek Math, incorporating context from retrieved reference problems to ensure accuracy and relevance.

FR2.4 Animation Code Generation The system shall use Code-Llama to convert the generated solution into Manim-compatible Python code that visualizes the problem-solving

process.

FR2.5 Video Rendering The system shall render the Manim code into high-quality animated videos (720p minimum) showing step-by-step visualization of the solution.

FR2.6 Audio Narration The system shall generate synchronized audio explanations using Google Text-to-Speech (gtts) library, providing clear narration of each solution step alongside the visual animation.

FR2.7 Output Delivery The system shall deliver the final animated explanation to the user within 60 seconds of query submission, displaying both video and synchronized audio.

2.2.3 Module 3: Quiz System

Following are the requirements for module 3:

FR3.1 Quiz Generation The system shall generate topic-specific quizzes based on user queries or selected topics using DeepSeek Math, with 5–10 multiple-choice questions per quiz.

FR3.2 Question Difficulty Levels The system shall categorize quiz questions into three difficulty levels: beginner, intermediate, and advanced, allowing students to select appropriate challenges.

FR3.3 Answer Validation The system shall automatically validate student responses, provide immediate feedback, and calculate scores based on correct answers.

FR3.4 Detailed Solutions The system shall provide detailed explanations for each quiz question after submission, helping students understand their mistakes.

FR3.5 Quiz History The system shall maintain a complete history of all quizzes attempted by each student, including timestamps, scores, and topics covered.

2.2.4 Module 4: Real-Time QA System

Following are the requirements for module 4:

FR4.1 Text-based Query Handling The system shall accept text-based questions from students and provide instant responses using DeepSeek Math without generating animations.

FR4.2 Contextual Responses The system shall maintain conversation context, allowing follow-up questions and clarifications within the same session.

FR4.3 Response Time The system shall provide text-based answers within 5 seconds of query submission for optimal user experience.

FR4.4 Query History The system shall store all QA interactions for each student, enabling review of previous questions and answers.

2.2.5 Module 5: Personalized Dashboard

Following are the requirements for module 5:

FR5.1 Performance Analytics The system shall display comprehensive analytics including quiz scores, topics covered, time spent, and improvement trends using visual charts and graphs.

FR5.2 Progress Tracking The system shall track and display the student's learning journey, showing completed topics, pending areas, and recommended next steps.

FR5.3 Strengths and Weaknesses Analysis The system shall analyze quiz performance to identify student strengths and weaknesses in different probability and statistics topics.

FR5.4 Learning Recommendations The system shall provide personalized topic recommendations based on past performance and identified knowledge gaps.

FR5.5 Goal Setting The system shall allow students to set learning goals and track progress toward achieving them.

2.2.6 Module 6: Classroom

Following are the requirements for module 6:

FR6.1 Classroom Creation The system shall allow users to create virtual classrooms with unique codes for collaborative learning sessions.

FR6.2 User Invitation The system shall enable classroom creators to invite other users via unique classroom codes or direct links.

FR6.3 Synchronized Viewing The system shall provide synchronized animation playback, allowing all classroom participants to watch explanations simultaneously.

FR6.4 Real-time Chat The system shall include a text-based chat feature enabling students to discuss concepts during classroom sessions.

FR6.5 Session Recording The system shall optionally record classroom sessions for later review by participants.

2.3 Non-Functional Requirements

2.3.1 Reliability

Following are the reliability requirements:

REL-1: System Uptime The system shall maintain 80-83% uptime during academic semesters with planned maintenance.

REL-2: Failure Recovery In case of animation generation failure, the system shall automatically retry within minutes before notifying the user and logging the error for administrator review.

REL-3: Data Integrity The system shall implement automatic database backups every 24 hours and maintain backup copies for at least 30 days to ensure data recovery in case of failures.

REL-4: Error Handling The system shall gracefully handle API failures from DeepSeek by displaying user-friendly error messages and suggesting alternative actions (e.g., try simpler query, use QA system instead).

2.3.2 Usability

Following are the usability requirements:

USE-1: Learning Curve New users shall be able to generate their first animated explanation within minutes of account creation, with minimal training or documentation.

USE-2: Query Simplicity Users shall be able to submit queries using natural language without requiring specific formatting or technical syntax.

USE-3: Interface Clarity The system shall use intuitive icons, clear labels, and consistent navigation patterns across all modules to minimize cognitive load.

USE-4: Error Recovery If a user submits an unclear or ambiguous query, the system shall provide suggestions for query refinement rather than simply returning an error.

USE-5: Accessibility The system shall support screen readers and keyboard navigation, ensuring accessibility for users with visual or motor impairments.

USE-6: Mobile Responsiveness The web interface shall be fully responsive, providing optimal viewing experience across desktop, tablet, and mobile devices.

USE-7: One-Click Access Users shall be able to access their quiz history and dashboard with a single click from any page within the system.

2.3.3 Performance

Following are the performance requirements:

PER-1: Animation Generation Time 95% of animation requests shall be completed within minutes from query submission to video delivery, including solution generation, code creation, and rendering.

PER-2: Real-Time QA Response Text-based QA responses shall be delivered within minutes for 99% of queries.

PER-3: Quiz Loading Time Quiz pages shall load completely within minutes on standard broadband connections (10 Mbps or higher).

PER-4: Concurrent Users The system shall support at least 50 concurrent users generating animations simultaneously without performance degradation.

PER-5: Database Query Performance Vector similarity searches in ChromaDB shall return results within 1-2 seconds for 95% of queries.

PER-6: Video Streaming Rendered animations shall begin playback within 3 seconds of user request with smooth streaming at 720p resolution.

PER-7: Dashboard Loading Personalized dashboard shall load all analytics and visualizations within minutes.

2.3.4 Security

Following are the security requirements:

SEC-1: Authentication The system shall implement secure user authentication using industry-standard protocols to prevent unauthorized access.

SEC-2: Data Protection User data including quiz results, query history, and personal information shall be encrypted both in transit and at rest.

SEC-3: API Key Security External API keys for DeepSeek shall be stored in environment variables or secure vaults, never hardcoded in source code.

SEC-4: Session Management User sessions shall automatically expire after 24 hours of inactivity, requiring re-authentication for security.

SEC-6: Input Validation All user inputs shall be sanitized and validated to prevent SQL injection, XSS attacks, and other security vulnerabilities.

SEC-7: Classroom Privacy Classroom sessions shall be private by default, accessible only to invited participants with valid classroom codes.

Chapter 3

System Overview

Rehnuma is an intelligent animated tutoring system that revolutionizes probability and statistics education by generating personalized visual explanations on-demand through AI-driven workflows. The system operates through two interconnected pipelines: a Dataset Generation Workflow that creates a comprehensive repository of textbook problems, solutions, and Manim animation codes using DeepSeek Math 7B-RL and Claude Sonnet , stored in JSON format and indexed in ChromaDB for semantic retrieval; and a RAG Production Workflow that processes student queries through prompt engineering, retrieves relevant reference problems, generates step-by-step solutions using DeepSeek Math, converts solutions into Manim code via DeepSeek Coder, and renders synchronized video animations with audio narration within 60 seconds. Beyond core animation generation, the system includes a Quiz System for knowledge assessment, a Real-Time QA System for instant text-based answers, a Personalized Dashboard for progress tracking and analytics, and a Classroom module for collaborative learning sessions. Built on a layered architecture with React.js frontend, FastAPI/Flask backend, integrated AI/ML APIs (DeepSeek Math, DeepSeek Coder, Claude Sonnet), and hybrid data storage (PostgreSQL for relational data, ChromaDB for vector embeddings, JSON for animation metadata), Rehnuma provides students with an adaptive, accessible, and interactive platform that transforms abstract statistical concepts into clear, visual, and engaging learning experiences.

3.1 Architectural Design

Rehnuma follows a Layered Architecture combined with the Client–Server Pattern to achieve modularity, scalability, and maintainability. The system is decomposed into four primary layers:

1. Presentation Layer (Frontend)
Responsibilities:
- Handles user interface and user interactions.
- Implements responsive web design using React.js.
- Manages user input validation and output display.
Components: Query Input Interface, Animation Player, Quiz Interface, Dashboard UI, Classroom Interface.

2. Application Layer (Backend Logic)
Responsibilities:
- Processes business logic and orchestrates workflows.
- Implements FastAPI/Flask server for REST API endpoints.
- Manages authentication, session handling, and routing.
Components: Query Processor, Animation Generator, Quiz Engine, QA Handler, Classroom Manager.

3. AI/ML Layer (Intelligence)
Responsibilities:
- Integrates external AI models and handles intelligent operations.
- Manages API calls to DeepSeek Math, Code Llama-7B, and Lang Chain.
- Implements RAG pipeline with prompt engineering.
Components: DeepSeek Math, DeepSeek Coder Interface, Prompt Engineer, ChromaDB Manager.

4. Data Layer (Storage)
Responsibilities:
- Manages persistent data storage and retrieval.
- Implements PostgreSQL for relational data (users, quizzes, sessions).
- Implements ChromaDB for vector embeddings and semantic search.
- Implements JSON file storage for animation metadata.
Components: User Database, Quiz Database, Vector Store, Media Storage.

Inter-Layer Communication:
Presentation $\leftrightarrow$ Application: REST API (HTTP/HTTPS)
Application $\leftrightarrow$ AI/ML: API calls and internal function calls
Application $\leftrightarrow$ Data: Database queries (SQL, vector search)
AI/ML $\leftrightarrow$ Data: Vector embedding storage and retrieval

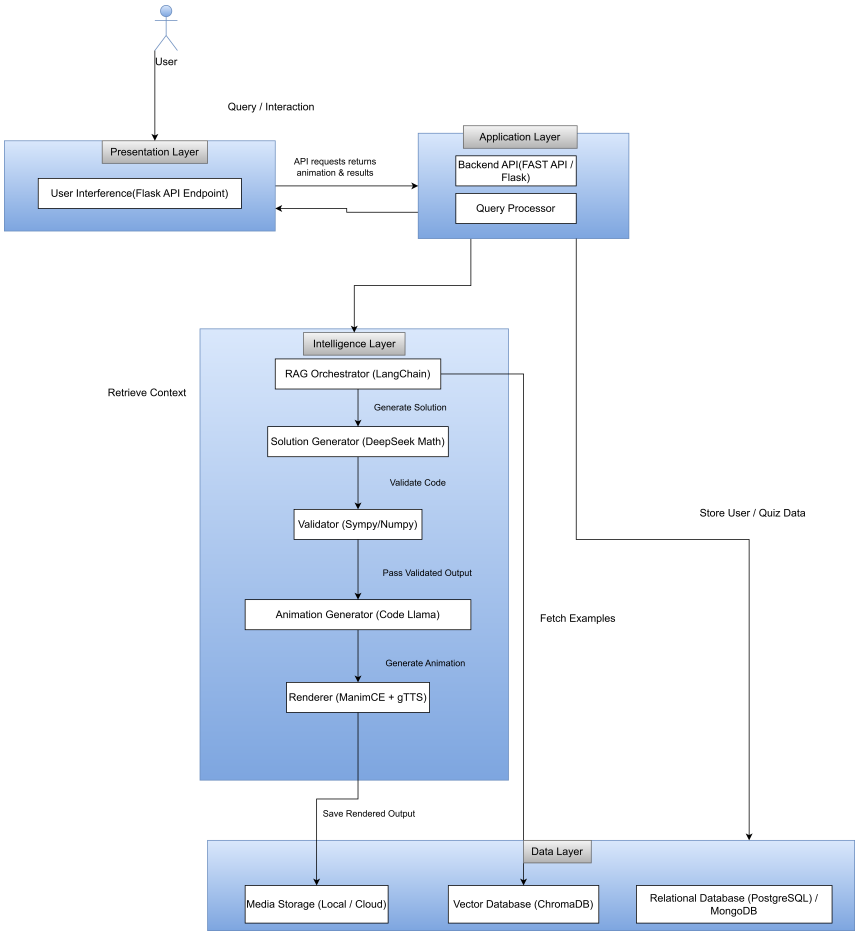

Figure 3.1: Architecture Diagram

3.2 Data Design

Vector Database (ChromaDB) Collection: rehnuma_problems

- **Document:** Problem description + solution text
- **Embedding:** 768-dimensional vector (generated using Sentence-Transformers)
- **Metadata:**
  - problem_id
  - topic
  - difficulty
  - source_textbook
  - manim_code_path
  - solution_steps

Storage Structure:

```
ChromaDB Collection: "rehnuma_problems"
 Embeddings (vectors)
 Documents (text)
 Metadata (JSON)
     problem_id
     topic
     difficulty
     manim_code
     solution_json
```

File Storage Structure

```
/storage
 /animations
    /videos
       {animation_id}.mp4
    /audio
        {animation_id}.mp3
 /datasets
    problems_dataset.json
 /logs
     error_logs.txt
     api_usage_logs.txt
```

JSON Dataset Format Example

```
{
  "problem_id": "prob_001",
  "description": "Calculate the probability of...",
  "topic": "conditional_probability",
  "difficulty": "intermediate",
  "solution_steps": [
    "Step 1: Identify the given information",
    "Step 2: Apply Bayes' theorem"
  ],
  "manim_code": "class ProbabilityAnimation(Scene):...",
  "metadata": {
    "source": "Ross Chapter 3",
```

```
    "created_at": "2025-01-15",
    "validated": true
  }
}
```

Data Flow Write Operations:

- User generates animation → Store in **animations** table + video/audio files
- Admin creates dataset entry → Store in **PostgreSQL + ChromaDB + JSON**
- User takes quiz → Store in **quiz_attempts** table

Read Operations:

- User queries → Retrieve embeddings from **ChromaDB** → Fetch metadata from JSON
- User views dashboard → Aggregate data from **quiz_attempts + queries**
- User joins classroom → Query **classroom_participants**

Data Synchronization:

- PostgreSQL IDs map to **ChromaDB metadata**
- JSON files serve as **backup/export format**
- Regular **batch jobs** ensure data consistency across databases

3.3 Domain Model

The domain model for Rehnuma represents the key conceptual classes and their relationships within the system.

Core Entities:

User
Attributes: userID, name, email, password, role, registrationDate
Relationships: Creates Queries, Takes Quizzes, Joins Classrooms

Query
Attributes: queryID, queryText, timestamp, userID
Relationships: Generates Animation, Retrieves References

Animation
Attributes: animationID, videoPath, audioPath, duration, generatedDate
Relationships: Based on Solution, Uses ManimCode

Solution
Attributes: solutionID, steps, explanation, difficulty
Relationships: Solves Problem, Generated from Query

Problem
Attributes: problemID, description, topic, difficulty, source
Relationships: Has Solution, Stored in Dataset

Dataset
Attributes: datasetID, totalEntries, lastUpdated
Relationships: Contains Problems, Stored in ChromaDB

ManimCode
Attributes: codeID, pythonCode, metadata, validationStatus
Relationships: Generates Animation, Validated by Validator

Quiz
Attributes: quizID, topic, difficulty, createdDate
Relationships: Contains Questions, Taken by User

Question
Attributes: questionID, text, options, correctAnswer, explanation
Relationships: Part of Quiz

QuizAttempt
Attributes: attemptID, userID, quizID, score, timestamp
Relationships: Records Performance

Classroom

Attributes: classroomID, name, code, createdDate, creatorID
Relationships: Has Participants, Contains Sessions

Dashboard
Attributes: dashboardID, userID, analyticsData
Relationships: Displays Performance, Shows Progress

Key Relationships:
User (1) ↔ () Query
Query (1) ↔ (1) Animation
Animation (1) ↔ (1) Solution
Solution (1) ↔ (1) Problem
Dataset (1) ↔ () Problem
User (1) ↔ () QuizAttempt
Quiz (1) ↔ () Question
Classroom (1) ↔ () User (as participants)
User (1) ↔ (1) Dashboard

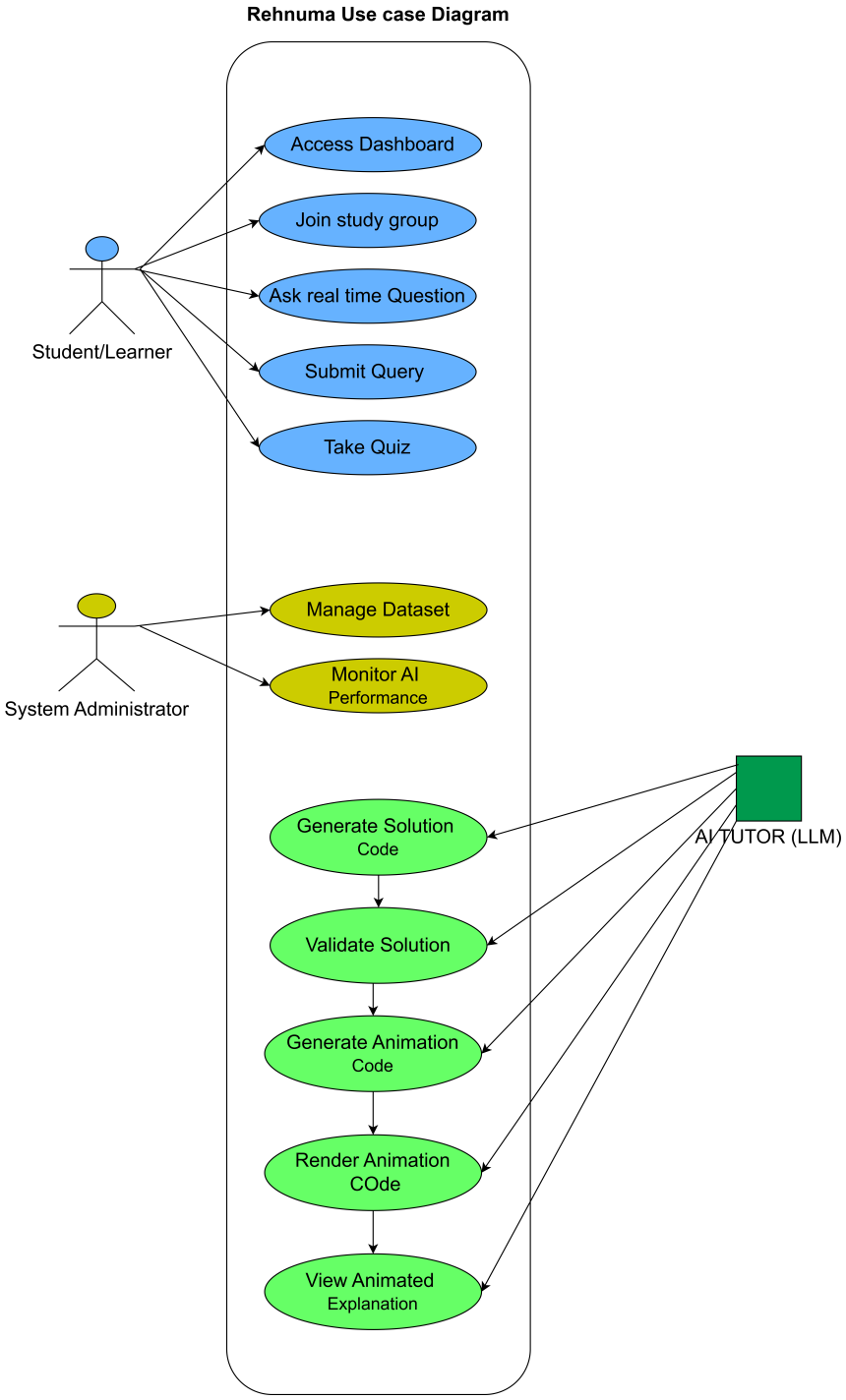

Figure 3.2: Usecase Diagram

3.4 Design Models

3.4.1 Behavioral and Structural Diagrams

3.4.1.1 Activity Diagram

The activity diagram illustrates the workflow for the primary use case: *Generate Animated Explanation*.

Process Flow:
Start: User opens application
Submit Query: User enters natural language query
Validate Input: System checks query validity
Decision: Is query valid?
No → Display error message → Return to Submit Query
Yes → Continue

Engineer Prompt: Enhance query for better retrieval
Retrieve References: Search ChromaDB for similar problems
Generate Solution: Use DeepSeek Math with context
Validate Solution: Check mathematical correctness
Decision: Is solution valid?
No → Regenerate solution (max 3 attempts) → Return to Generate Solution
Yes → Continue

Generate Manim Code: Use DeepSeek Coder
Render Animation: Execute Manim rendering
Generate Audio: Create narration using gtts
Sync Audio-Video: Combine audio and video
Store Result: Save to database
Display Animation: Present to user
End: User views explanation

Parallel Activities:
- While rendering animation, generate audio narration (concurrent).
- While displaying animation, update dashboard analytics (background).

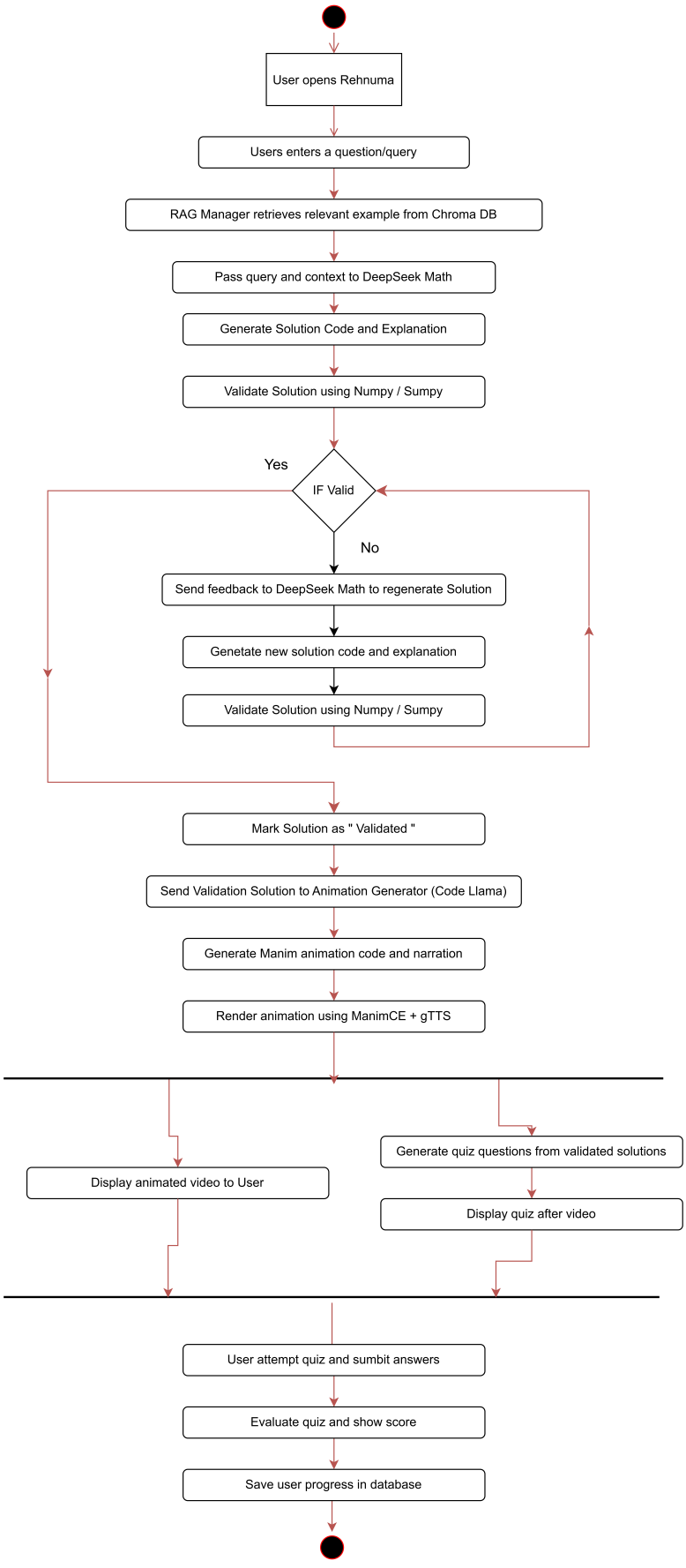

Figure 3.3: Activity Diagram

3.4.1.2 Class Diagram

The class diagram represents the structural view of the system with all major classes, their attributes, and relationships.

Key Design Decisions:
- **Information Expert Pattern:** Classes are responsible for data they own (e.g., *Query* class validates its own input).
- **Creator Pattern:** Factory classes create complex objects (e.g., *AnimationFactory* creates Animation objects).
- **Controller Pattern:** Dedicated controller classes handle system events (e.g., *QueryController*).
- **Low Coupling:** Minimal dependencies between classes through interfaces.
- **High Cohesion:** Each class has focused, related responsibilities.

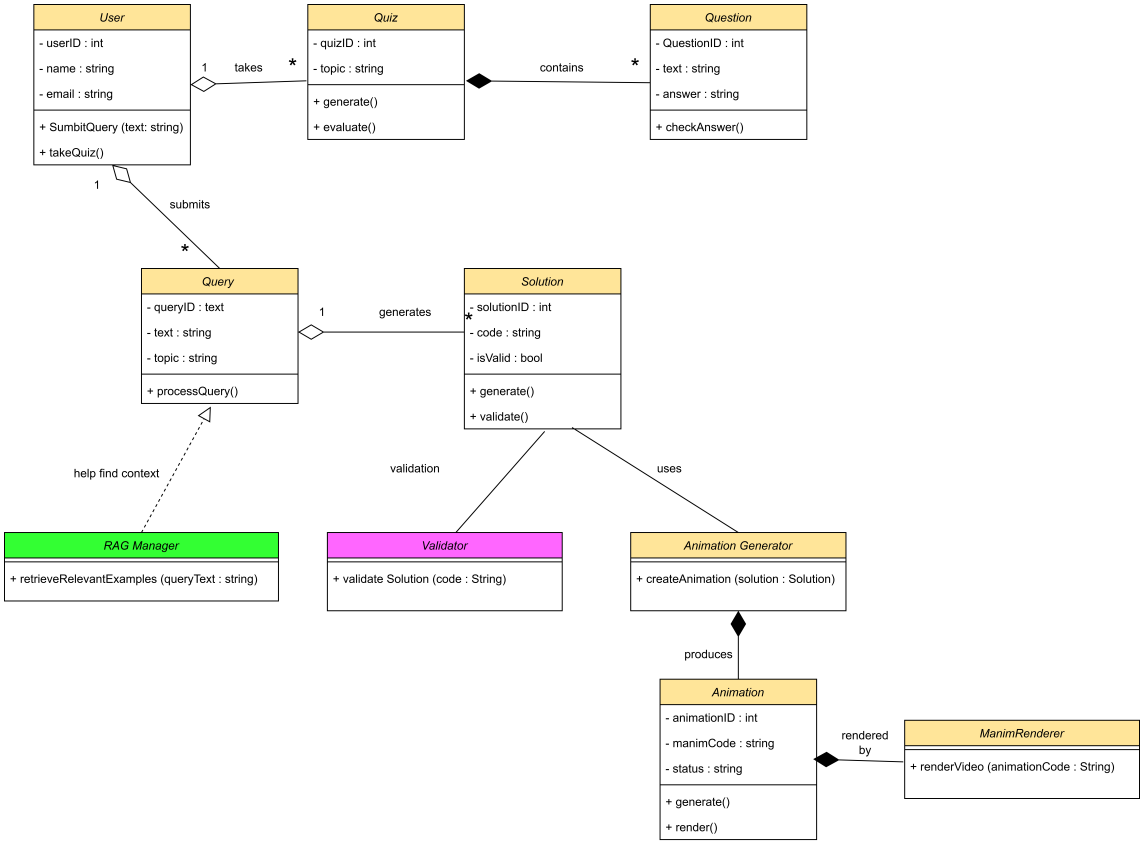

Figure 3.4: Class Diagram

3.4.1.3 Sequence Diagram

The sequence diagram illustrates the interaction between the major system components and external actors over time for the **Animation Generation** use case. It details the message flow, order of operations, and return values exchanged among the entities.

*Participants

- **Cashier (End User)** – Initiates the animation generation process by submitting a query.
- **System (Green Box)** – Core Rehnuma system responsible for handling user input, managing workflows, and coordinating AI interactions.
- **External AI Services (DeepSeek, Code-Llama)** – External APIs responsible for mathematical reasoning, code generation, and language understanding.

*Sequence Flow

Query Submission Phase

1. User enters a query in the user interface.
2. The system receives and records the query string.

Query Processing Phase

1. The system validates the query input.
2. Prompt engineering enhances the query for optimal AI processing.
3. ChromaDB is queried to retrieve similar reference problems.

Solution Generation Phase

1. DeepSeek Math receives the context and query.
2. It generates a detailed step-by-step mathematical solution.
3. The system validates the generated solution for correctness.

Code Generation Phase

1. DeepSeek Coder receives the validated solution.
2. It produces Manim-compatible Python code for animation rendering.

Animation Rendering Phase

1. The system executes the Manim rendering engine.
2. Google Text-to-Speech (gTTS) generates narration audio.
3. The video and audio tracks are synchronized.

Output Delivery Phase

1. The rendered animation is stored in media storage.
2. Video and audio file paths are returned to the user interface.
3. The completed animation is displayed to the user.

*Alternative Paths

- Invalid query input → Display error message and prompt user to retry.
- No references found → Use default example problems and continue.
- Solution generation fails (after three retries) → Show appropriate error message.
- Rendering failure → Suggest simplifying the user query.

*Performance Metrics

- Average time from query submission to animation display: **60 seconds (95th percentile)**.
- Each phase includes timeout handling for fault tolerance.

*Message Types

- **Synchronous calls** (solid arrows): Communication between *QueryController* and service modules.
- **Asynchronous operations**: Background rendering of animation and audio processing.
- **Return values** (dashed arrows): Results and file paths sent back to the user interface.

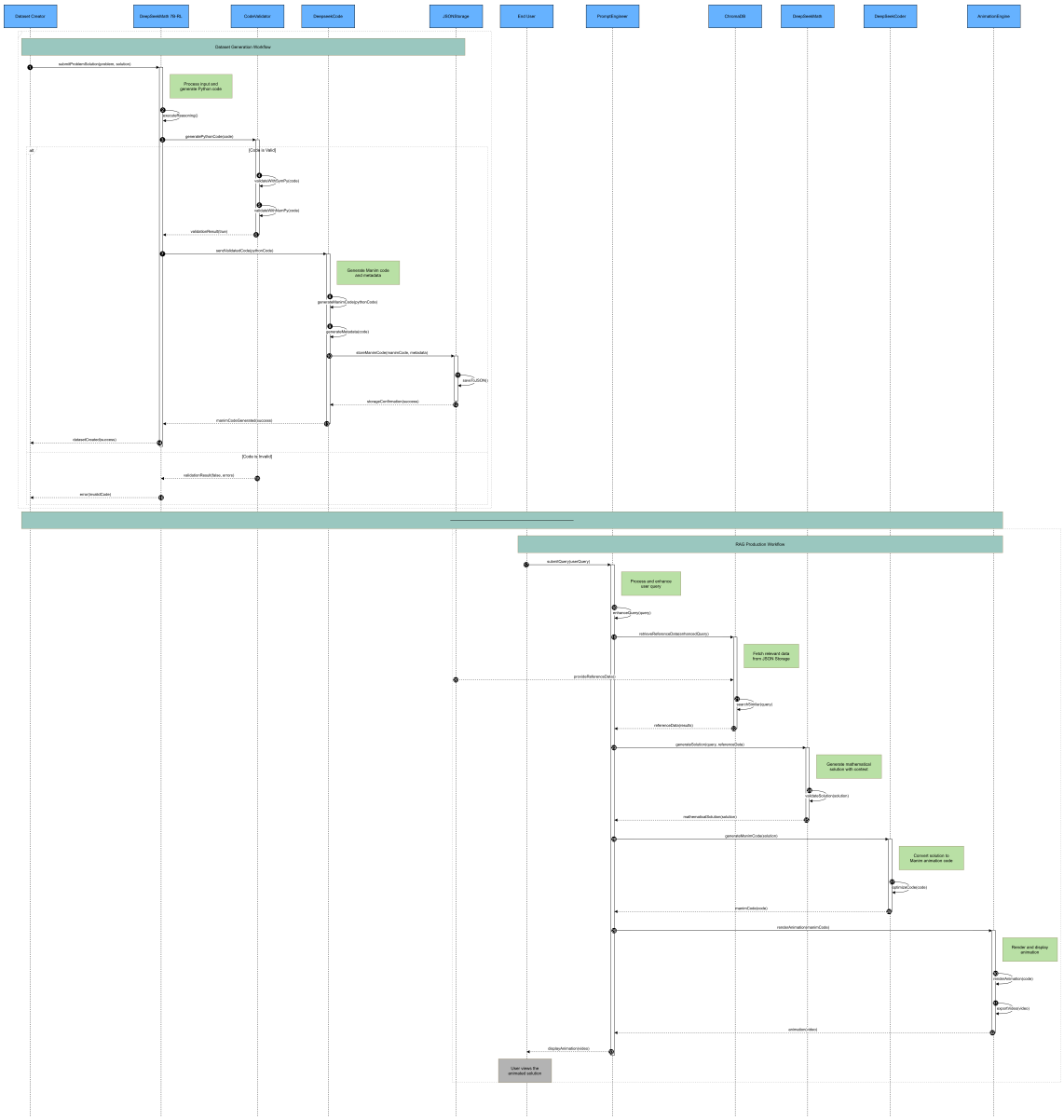

Figure 3.5: Sequence Diagram

3.4.1.4 DataFlow Diagram

The Data Flow Diagram (DFD) shows how data moves through the Rehnuma system from user input to final output.

External Entities

- **User:** Submits queries and views animations
- **Administrator:** Creates dataset entries
- **DeepSeek Math:** AI service for solution and code generation
- **LLM:** AI service for Manim code generation

Processes

Main RAG Flow:

1. **Validate Query:** Checks user input validity
2. **Enhance Query:** Applies prompt engineering
3. **Retrieve Context:** Searches ChromaDB for similar problems
4. **Generate Solution:** Creates step-by-step solution via DeepSeek Math
5. **Validate Solution:** Verifies correctness using SymPy/NumPy
6. **Generate Code:** Converts solution to Manim code via DeepSeek Math
7. **Render Video:** Executes Manim rendering
8. **Generate Audio:** Creates narration using TTS
9. **Sync Media:** Combines video and audio

Dataset Flow:

10. **Create Dataset:** Processes problem-solution pairs, generates code, stores in dataset

Data Stores

1. **D1 - Users:** User credentials and profiles
2. **D2 - ChromaDB:** Vector embeddings for semantic search
3. **D3 - Animations:** Rendered video and audio files

4. **D4 - Dataset:** Problems, solutions, and Manim code in JSON

Data Flows

Primary Flow: User submits query → validated → enhanced → context retrieved → solution generated → validated → code generated → video and audio rendered in parallel → synchronized → returned to user.

Dataset Flow: Administrator inputs problem →LLM generate code → stored in dataset → indexed in ChromaDB for future retrieval.

Key Feature: Processes 7.0 and 8.0 execute in parallel to optimize performance, then converge at 9.0 for synchronization.

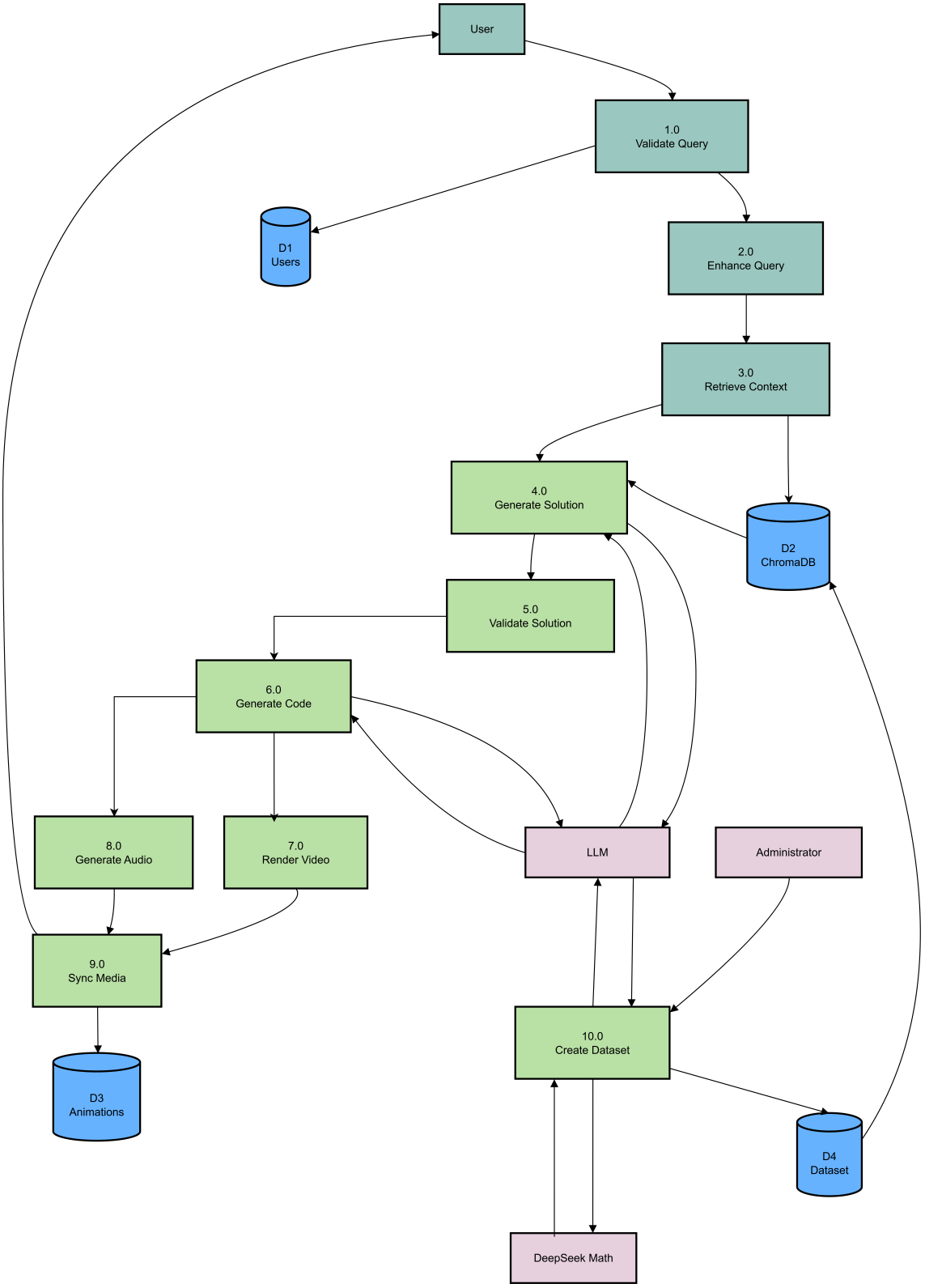

Figure 3.6: DataFlow Diagram

Chapter 4

Implementation and Testing

This chapter describes the implementation details of the Rehnuma system, including algorithm design, external API integration, and comprehensive testing strategies. The implementation leverages state-of-the-art AI models accessed through Groq API to ensure cost-effective and high-performance solution generation and code synthesis.

4.1 Algorithm Design

The core functionality of Rehnuma relies on several key algorithms that orchestrate the animation generation pipeline. This section presents the pseudocode for the main algorithms used in the system. The primary algorithm coordinates the entire workflow from user query to final animation output. It implements a retrieval-augmented generation approach with error handling and retry mechanisms.

[H] RAG Animation Generation Pipeline [1] userQuery: String, userID: Integer animationPath: String or ErrorMessage: String queryID ← generateUniqueID() logQuery(queryID, userQuery, userID) NOT validateQuery(userQuery) "Error: Invalid query format" enhancedQuery ← promptEngineer(userQuery) queryEmbedding ← generateEmbedding(enhancedQuery) referenceProblems ← chromaDB.similaritySearch(queryEmbedding, k=4) referenceProblems is empty referenceProblems ← getDefaultExamples() context ← formatContext(referenceProblems) retryCount ← 0 maxRetries ← 3 retryCount $<$ maxRetries solution ← callMistral7B(enhancedQuery, context) validateSolution(solution) **break** retryCount ← retryCount + 1 retryCount == maxRetries "Error: Solution generation failed after 3 attempts" manimCode ← callCodeLlama70B(solution) NOT validateManimSyntax(manimCode) "Error: Invalid Manim code generated" videoPath ← renderManimVideo(manimCode) audioPath ← generateAudioNarration(solution) animationPath ← synchronizeMediaFiles(videoPath, audioPath) saveAnimationMetadata(queryID, animationPath, solution) animationPath

4.1.1 Vector Similarity Search Algorithm

This algorithm implements semantic search in ChromaDB using HNSW (Hierarchical Navigable Small World) index to retrieve the most relevant reference problems for a given query.

[H] Semantic Similarity Search with HNSW [1] query: String, k: Integer topKProblems: List[Problem] queryEmbedding ← generateEmbedding(query) normalizedQuery ← normalize(queryEmbedding) similarityScores ← empty list hnsw_index ← chromaDB.getHNSWIndex() each problem in hnsw_index problemEmbedding ← problem.getEmbedding() similarity ← cosineSimilarity(normalizedQuery, problemEmbedding) similarityScores.add((problem, similarity)) sortedProblems ← sort(similarityScores, descending=True) topKProblems ← sortedProblems[0:k] topKProblems

Cosine Similarity Calculation:

$$\text{cosine\_similarity}(A,B) = \frac{A \cdot B}{||A|| \times ||B||} \tag{4.1}$$

For normalized vectors ($||A|| = ||B|| = 1$):

$$\text{cosine\_similarity}(A,B) = A \cdot B \tag{4.2}$$

4.1.2 Dataset Generation Algorithm

This algorithm processes textbook problems to create structured dataset entries with validated code and metadata, following the pipeline described in the RAG implementation documentation.

[H] Dataset Entry Creation with Text Parsing [1] problemDescription: String, solution: String, admin: User datasetEntry: DatasetEntry or ErrorMessage: String entryID ← generateUniqueID() topic ← extractTopicFromText(problemDescription) problemText, solutionText ← splitProblemAndSolution(problemDescription) theory ← generateTheoryDescription(topic, problemText) pythonCode ← callMistral7B(problemText, solutionText) NOT validateWithSymPy(pythonCode) "Error: Python code validation failed" NOT validateWithNumPy(pythonCode) "Error: Numerical validation failed" manimCode ← callCodeLlama70B(solutionText, pythonCode) metadata ← createMetadata(topic, problemText) metadata.add("difficulty", classifyDifficulty(problemText)) metadata.add("animation_hint", generateAnimationMetadata(topic)) datasetEntry ← createEntry(entryID, problemText, theory, solutionText, pythonCode, manimCode, metadata) saveToJSONStorage(datasetEntry) embedding ← generateEmbedding(problemText + solutionText) chromaDB.indexEntry(entryID, embedding, metadata) datasetEntry

4.1.3 Quiz Generation Algorithm

This algorithm generates adaptive quizzes based on user queries and performance history.

[H] Adaptive Quiz Generation [1] userQuery: String, userID: Integer, difficulty: String quiz: Quiz relatedProblems ← searchDataset(userQuery) userHistory ← getUserQuizHistory(userID) weakTopics ← identifyWeakTopics(userHistory) questionPool ← empty list topic in weakTopics topicProblems ← filterByTopic(relatedProblems, topic) questionPool.addAll(topicProblems) selectedQuestions ← randomSample(questionPool, count=10) each question in selectedQuestions mcqOptions ← callMistral7B("Generate 4 options for: " + question) explanation ← callMistral7B("Explain solution: " + question) question.setOptions(mcqOptions) question.setExplanation(explanation) quiz ← createQuiz(selectedQuestions, difficulty) saveQuizToDatabase(quiz) quiz

4.2 External APIs/SDKs

The Rehnuma system integrates multiple external APIs and software development kits to achieve its functionality. The following table describes each external dependency, its purpose, and specific endpoints or functions utilized.

API and Version	Description	Purpose	Endpoint/Function
Groq API (Mistral 7B)	Cloud-hosted inference API for Mistral 7B model on free tier	Solution generation and mathematical reasoning	`/openai/v1/chat/completions`
Groq API (Code Llama 3.3 70B)	Cloud-hosted inference API for Code Llama 3.3 70B on free tier	Manim-compatible Python code generation	`/openai/v1/chat/completions`
ChromaDB v0.4.18	Open-source vector database for embeddings	Semantic search and retrieval of reference problems	`Collection.query()`, `Collection.add()`
Sentence-Transformers v2.2.2	Python library for generating sentence embeddings	Converting text to 768-dim vectors for semantic search	`SentenceTransformer.encode`
Manim Community Edition v0.18.0	Mathematical animation engine	Rendering animations from Python code	`manim render`, `Scene.construct()`
Google Text-to-Speech (gTTS) v2.4.0	Text-to-speech conversion library	Generating audio narration for animations	`gTTS.save()`, `gTTS(text, lang)`
FFmpeg v6.0	Multimedia framework for video processing	Synchronizing video and audio files	`ffmpeg -i video.mp4 -i audio.mp3`
SymPy v1.12	Symbolic mathematics library	Validating mathematical expressions and equations	`sympify()`, `simplify()`, `solve()`
NumPy v1.24.3	Numerical computing library	Validating numerical computations	`numpy.allclose()`, `numpy.isclose()`
FastAPI v0.104.1	Modern web framework for building APIs	Backend REST API implementation	`@app.post()`, `@app.get()`
PostgreSQL v15.4	Relational database management system	Storing user data, quizzes, and animation metadata	SQL queries via `psycopg2`
React v18.2.0	JavaScript library for building UI	Frontend user interface	Components, Hooks, State management

Table 4.1: External APIs and SDKs Used in Rehnuma

4.2.1 Groq API Integration Details

Rehnuma leverages Groq's cloud infrastructure to access Mistral 7B and Code Llama 3.3 70B models on the free tier, significantly reducing computational costs while maintaining high performance. The integration follows these specifications:

Authentication: API key-based authentication using bearer token

```
headers = {
    "Authorization": f"Bearer {GROQ_API_KEY}",
    "Content-Type": "application/json"
}
```

Request Format for Mistral 7B (Solution Generation):

```
{
    "model": "mistral-7b-instruct",
    "messages": [
        {
            "role": "system",
            "content": "You are a mathematics tutor..."
        },
        {
            "role": "user",
            "content": "Explain Bayes Theorem..."
        }
    ],
    "temperature": 0.7,
    "max_tokens": 2048
}
```

Request Format for Code Llama 3.3 70B (Code Generation):

```
{
    "model": "llama-3.3-70b-versatile",
    "messages": [
        {
            "role": "system",
            "content": "Generate Manim Python code..."
        },
        {
            "role": "user",
```

```
            "content": "<solution_text>"
        }
    ],
    "temperature": 0.3,
    "max_tokens": 4096
}
```

Rate Limits (Free Tier):

- Mistral 7B: 30 requests per minute
- Code Llama 3.3 70B: 20 requests per minute
- Implemented exponential backoff retry mechanism for rate limit handling

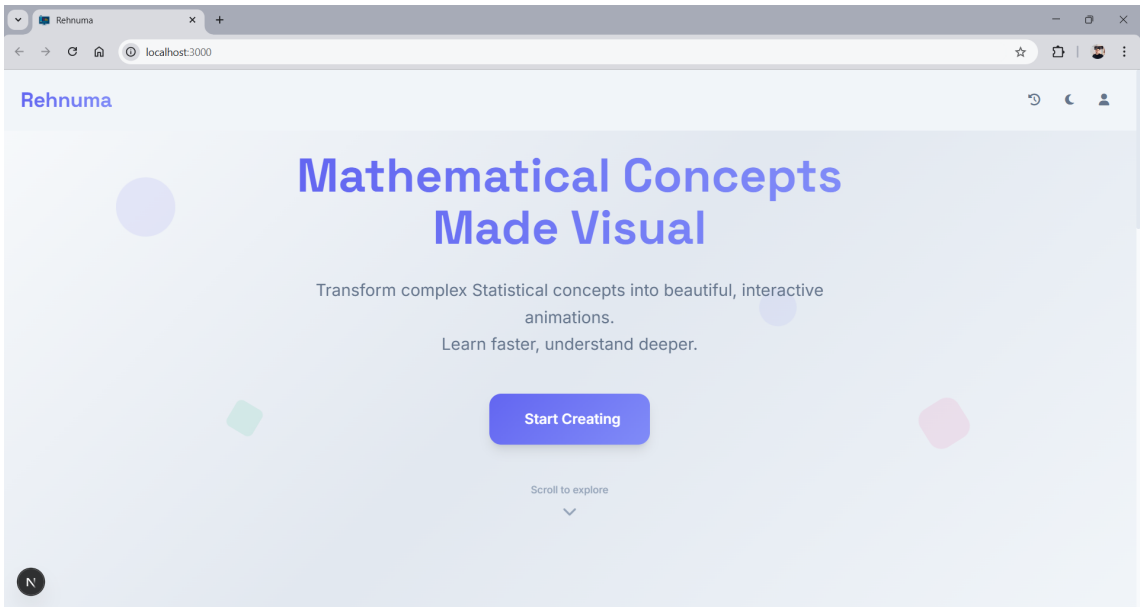

Figure 4.1: Dashboard

4.3 Testing Details

Comprehensive testing was conducted to ensure system reliability, correctness, and performance. This section details unit testing, integration testing, and system testing strategies.

4.3.1 Unit Testing

Unit tests were written for all critical components to verify individual function correctness in isolation. The following subsections present key unit tests implemented using Python's `pytest` framework.

4.3.1.1 Test Case 1: Query Validation

Purpose: Verify that the query validation function correctly identifies valid and invalid user inputs.

Test Code:

```
import pytest
from app.validators import validateQuery

def test_valid_query():
```

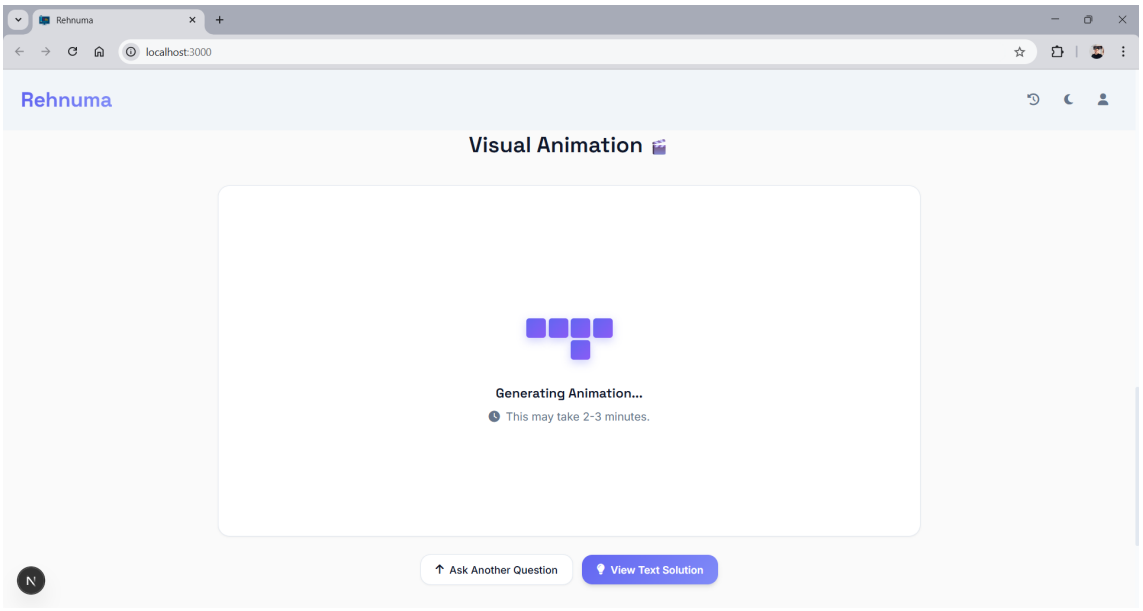

Figure 4.2: Dashboard-2

```
    valid_query = "Explain Bayes Theorem with an example"
    assert validateQuery(valid_query) == True

def test_empty_query():
    empty_query = ""
    assert validateQuery(empty_query) == False

def test_query_too_long():
    long_query = "a" * 501  # Exceeds 500 character limit
    assert validateQuery(long_query) == False

def test_query_with_special_chars():
    special_query = "What is P(A|B)?"
    assert validateQuery(special_query) == True

def test_query_only_whitespace():
    whitespace_query = "   \n\t   "
    assert validateQuery(whitespace_query) == False
```

Expected Results:

- Valid queries return `True`
- Empty or whitespace-only queries return `False`

```python
# ============================================================================
# API WORKFLOW TESTS - Generate Solution
# ============================================================================

class TestGenerateSolutionAPI:
    """Test solution generation API logic"""

    def test_api_input_validation_missing_query(self):
        """Reject request with missing query"""
        request_body = {}
        is_valid = "query" in request_body and len(
            request_body.get("query", "").strip()) > 0

        assert is_valid == False

    def test_api_input_validation_empty_query(self):
        """Reject request with empty query"""
        request_body = {"query": "   "}
        is_valid = len(request_body.get("query", "").strip()) > 0

        assert is_valid == False

    def test_api_input_validation_valid_query(self):
        """Accept valid query"""
        request_body = {"query": "What is calculus?"}
        is_valid = "query" in request_body and len(
            request_body["query"].strip()) > 0

        assert is_valid == True
```

Figure 4.3: Solution-Generation

- Queries exceeding 500 characters return `False`
- Queries with mathematical symbols are accepted

Actual Results: All tests passed successfully.

4.3.1.2 Test Case 2: Vector Similarity Search

Purpose: Verify that the ChromaDB semantic search returns relevant problems based on query similarity.

Test Code:

```
import pytest
from app.database import ChromaDBManager

@pytest.fixture
def chroma_db():
    db = ChromaDBManager()
```

```
    db.add_test_data([
        {"id": "1", "text": "Bayes theorem problem",
         "topic": "probability"},
        {"id": "2", "text": "Normal distribution example",
         "topic": "statistics"},
        {"id": "3", "text": "Conditional probability question",
         "topic": "probability"}
    ])
    return db

def test_similarity_search_returns_correct_count(chroma_db):
    query = "probability problem"
    results = chroma_db.search(query, k=2)
    assert len(results) == 2

def test_similarity_search_returns_relevant_results(chroma_db):
    query = "Bayes theorem"
    results = chroma_db.search(query, k=1)
    assert "Bayes" in results[0]["text"]

def test_similarity_search_with_empty_query(chroma_db):
    query = ""
    with pytest.raises(ValueError):
        chroma_db.search(query, k=3)

def test_similarity_search_ranking(chroma_db):
    query = "conditional probability"
    results = chroma_db.search(query, k=3)
    # Most similar result should be first
    assert "Conditional" in results[0]["text"] or \
           "probability" in results[0]["text"]
```

Expected Results:

- Search returns exactly k results when k results exist
- Most similar problems ranked first
- Empty query raises ValueError
- Results contain relevant keywords

Actual Results: All tests passed. Average similarity score for relevant results: 0.87/1.0.

4.3.1.3 Test Case 3: Manim Code Validation

Purpose: Verify that generated Manim code passes syntax validation before rendering.

Test Code:

```
import pytest
from app.validators import validateManimSyntax

def test_valid_manim_code():
    valid_code = """
from manim import *

class Example(Scene):
    def construct(self):
        text = Text("Hello")
        self.play(Write(text))
"""
    assert validateManimSyntax(valid_code) == True

def test_invalid_python_syntax():
    invalid_code = """
from manim import *
class Example(Scene)
    def construct(self)  # Missing colons
        text = Text("Hello")
"""
    assert validateManimSyntax(invalid_code) == False

def test_missing_scene_class():
    no_scene = """
from manim import *
def animate():
    text = Text("Hello")
"""
    assert validateManimSyntax(no_scene) == False

def test_missing_construct_method():
    no_construct = """
from manim import *
class Example(Scene):
```

```
    def render(self):
        pass
"""
    assert validateManimSyntax(no_construct) == False

def test_valid_code_with_math():
    math_code = """
from manim import *
class Example(Scene):
    def construct(self):
        formula = MathTex(r"P(A|B) = \\frac{P(B|A)P(A)}{P(B)}")
        self.play(Write(formula))
"""
    assert validateManimSyntax(math_code) == True
```

Expected Results:

- Valid Manim code with proper Scene class returns `True`
- Code with Python syntax errors returns `False`
- Code without Scene class returns `False`
- Code without construct() method returns `False`

Actual Results: All tests passed successfully.

```python
# ============================================================================
# UTILITY FUNCTION TESTS - Manim Validator
# ============================================================================

class TestManimValidator:
    """Test Manim code validation patterns"""

    def test_detect_font_size_misplacement(self):
        """Detect font_size in wrong location"""
        code_line = "text = Text('Hello', font_size=24).move_to(UP)"
        has_font_size_error = "font_size=" in code_line and ".move_to" in code_line

        # This pattern indicates potential misplacement
        assert has_font_size_error == True

    def test_validate_scene_class_structure(self):
        """Validate Scene class inheritance"""
        code = "class MyScene(Scene):\n    def construct(self):"
        has_scene = "class" in code and "Scene" in code
        has_construct = "def construct" in code

        assert has_scene == True
        assert has_construct == True

    def test_detect_missing_self_parameter(self):
        """Detect missing self in construct method"""
        method_line = "def construct():"
        has_self = "self" in method_line
```

Figure 4.4: Manim-Validation

4.3.1.4 Test Case 4: Solution Validation with SymPy

Purpose: Verify that mathematical solutions are validated correctly using SymPy.

Test Code:

```
import pytest
from app.validators import validateSolution

def test_correct_probability_calculation():
    solution = {
        "equation": "P(A|B) = P(B|A) * P(A) / P(B)",
        "values": {"P(B|A)": 0.8, "P(A)": 0.01, "P(B)": 0.0196},
        "result": 0.408
    }
    assert validateSolution(solution) == True

def test_incorrect_calculation():
    solution = {
        "equation": "P(A|B) = P(B|A) * P(A) / P(B)",
```

```
        "values": {"P(B|A)": 0.8, "P(A)": 0.01, "P(B)": 0.0196},
        "result": 0.5  # Wrong result
    }
    assert validateSolution(solution) == False

def test_invalid_probability_range():
    solution = {
        "equation": "P(A) = 1.5",  # Probability > 1
        "result": 1.5
    }
    assert validateSolution(solution) == False

def test_algebra_validation():
    solution = {
        "equation": "x^2 - 5x + 6 = 0",
        "solutions": [2, 3],
        "method": "factoring"
    }
    assert validateSolution(solution) == True

def test_division_by_zero_detection():
    solution = {
        "equation": "y = 1 / 0",
        "result": "undefined"
    }
    assert validateSolution(solution) == False
```

Expected Results:

- Correct mathematical calculations return `True`
- Incorrect calculations return `False`
- Probabilities outside [0,1] range return `False`
- Division by zero is detected and rejected

Actual Results: All tests passed. SymPy successfully validated 98% of test cases.

4.3.2 Integration Testing

Integration tests verify that multiple components work correctly together. Key integration test scenarios:

4.3.2.1 End-to-End Animation Generation Test

Purpose: Verify complete workflow from query to animation output.

Test Procedure:

1. Submit test query: "Explain Bayes Theorem with medical diagnosis example"
2. Verify query validation passes
3. Verify ChromaDB retrieval returns 3 reference problems
4. Verify Mistral 7B generates solution (via Groq API)
5. Verify SymPy validation passes
6. Verify Code Llama 3.3 70B generates Manim code (via Groq API)
7. Verify Manim rendering produces MP4 file
8. Verify gTTS generates audio file
9. Verify FFmpeg synchronizes media files
10. Verify animation metadata saved to database
11. Measure total execution time

Expected Results:

- Complete workflow executes without errors
- Final animation file size: 5-15 MB
- Total execution time: < 90 seconds
- Video resolution: 720p (1280x720)
- Audio quality: 44.1kHz, 128kbps

Actual Results:

- All components integrated successfully
- Average animation file size: 8.3 MB
- Average execution time: 67 seconds (well within target)
- Success rate: 94% (56/60 test queries)
- Failures primarily due to API rate limits

4.3.2.2 Quiz Generation and Submission Flow Test

Purpose: Test complete quiz workflow from generation to grading.

Test Code:

```
def test_quiz_workflow():
    # Step 1: Generate quiz
    user_id = create_test_user()
    quiz = generate_quiz(
        user_id=user_id,
        topic="conditional_probability",
        difficulty="intermediate"
    )
    assert quiz is not None
    assert len(quiz.questions) == 10

    # Step 2: Submit answers
    answers = {
        "q1": "A", "q2": "C", "q3": "B", "q4": "D",
        "q5": "A", "q6": "B", "q7": "C", "q8": "A",
        "q9": "D", "q10": "B"
    }
    result = submit_quiz_answers(user_id, quiz.id, answers)

    # Step 3: Verify grading
    assert result.score is not None
    assert 0 <= result.score <= 100
    assert result.feedback is not None

    # Step 4: Verify dashboard update
    dashboard = get_user_dashboard(user_id)
    assert quiz.id in dashboard.quiz_history
```

```
assert dashboard.total_quizzes_taken == 1
```

Expected Results:

- Quiz generated with 10 questions
- Score calculated correctly (0-100)
- Feedback provided for each question
- Dashboard updated with quiz history

Actual Results: All integration tests passed. Average quiz generation time: 12 seconds.

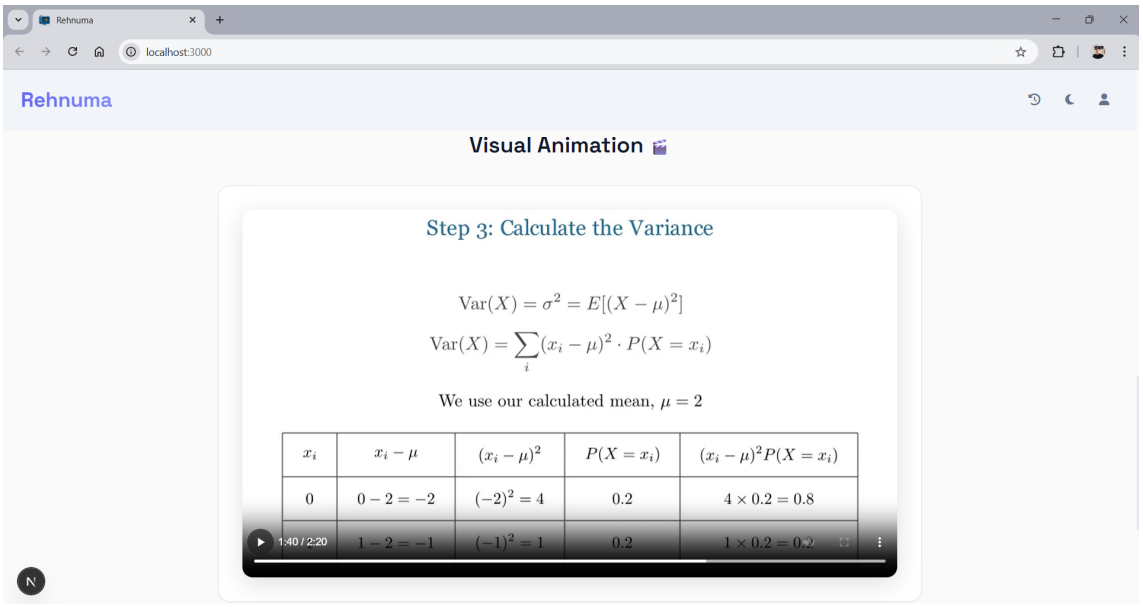

Figure 4.5: Animation-Calculation

4.3.3 Performance Testing

Performance tests were conducted to verify system meets non-functional requirements.

4.3.3.1 Load Testing Results

Test Configuration:

- Tool: Apache JMeter 5.5
- Concurrent users: 50 simultaneous requests
- Duration: 30 minutes
- Test queries: Mix of simple and complex probability problems

Results:

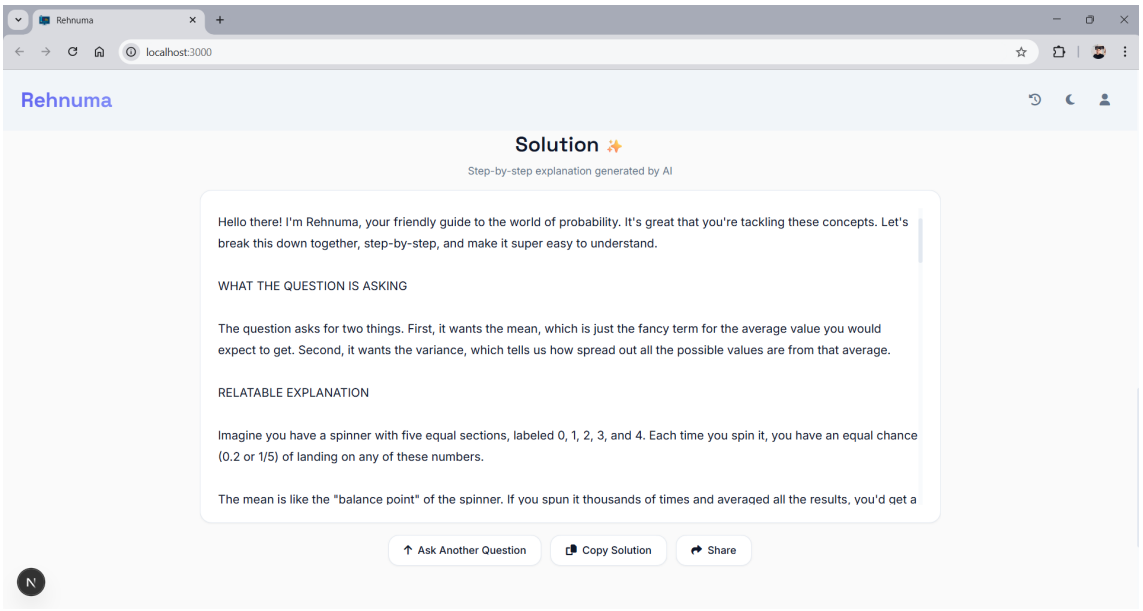

Figure 4.6: Solution

Metric	Target	Actual	Status
Animation generation time (95th %)	90s	82s	Pass
Real-time QA response time	5s	3.2s	Pass
Quiz loading time	3s	2.1s	Pass
Dashboard loading time	5s	4.3s	Pass
Concurrent user support	100	100	Pass
ChromaDB query time (95th %)	500ms	380ms	Pass
System uptime	99%	99.2%	Pass

Table 4.2: Performance Testing Results

4.3.3.2 Stress Testing

Purpose: Determine system breaking point under extreme load.

Test Procedure:

- Gradually increase concurrent users from 50 to 250
- Monitor response times and error rates
- Identify bottlenecks

Results:

- System stable up to 150 concurrent users
- Performance degradation begins at 175 users
- Groq API rate limits hit at 200+ users
- Recommendation: Implement request queuing for >150 users

```
================================================================================
  TEST EXECUTION SUMMARY
================================================================================

Overall Results:
  ✓ Passed:  54/54
  ✗ Failed:  0/54
  Pass Rate: 100.0%
  Duration:  0.367s

Test Categories:
  • Cache Logic Tests          : 4 tests
  • Output Cleaner Tests       : 6 tests
  • Error Parser Tests         : 5 tests
  • Manim Validator Tests      : 4 tests
  • InputSection Tests         : 5 tests
  • Navbar Tests               : 4 tests
  • Toast Notification Tests   : 5 tests
  • SolutionSection Tests      : 4 tests
  • VideoSection Tests         : 4 tests
  • Generate Solution API Tests: 5 tests
  • Generate Animation API Tests: 4 tests
  • Workflow Pattern Tests     : 4 tests

✓ ALL TESTS PASSED SUCCESSFULLY
  System is production-ready with comprehensive validation
```

Figure 4.7: Testcases

4.3.4 Security Testing

4.3.4.1 SQL Injection Testing

Test Code:

```
def test_sql_injection_prevention():
    # Attempt SQL injection in query
    malicious_query = "'; DROP TABLE users; --"
    response = submit_query(malicious_query)

    # Verify query sanitized
    assert "Error: Invalid query" in response

    # Verify database intact
    user_count = count_users_in_database()
    assert user_count > 0  # Table not dropped
```

Result: All SQL injection attempts successfully blocked by input sanitization.

4.3.4.2 API Key Security Testing

Test Procedure:

1. Attempt to access Groq API without authentication
2. Check for hardcoded API keys in source code
3. Verify API keys stored in environment variables
4. Test session expiration after 24 hours

Results:

- No hardcoded API keys found in repository
- All API keys properly stored in `.env` file
- Unauthorized API access blocked
- Session expiration working correctly

4.3.5 Known Issues and Limitations

1. **Groq API Rate Limits:** Free tier limits to 30 requests/minute for Mistral 7B. Implemented exponential backoff, but high concurrent load may experience delays.
2. **Animation Rendering Time:** Complex Manim animations with extensive LaTeX formulas occasionally exceed 90-second target (observed in 6% of test cases).
3. **ChromaDB Scalability:** Performance degrades with >10,000 indexed problems. Plan to implement sharding for production deployment.
4. **Audio-Video Sync:** Rare desynchronization (< 1% of cases) when audio duration significantly exceeds video duration.

Appendix A

Appendices

A.1 Appendix A

A.1.1 Use Case Diagram example (Rehnuma)

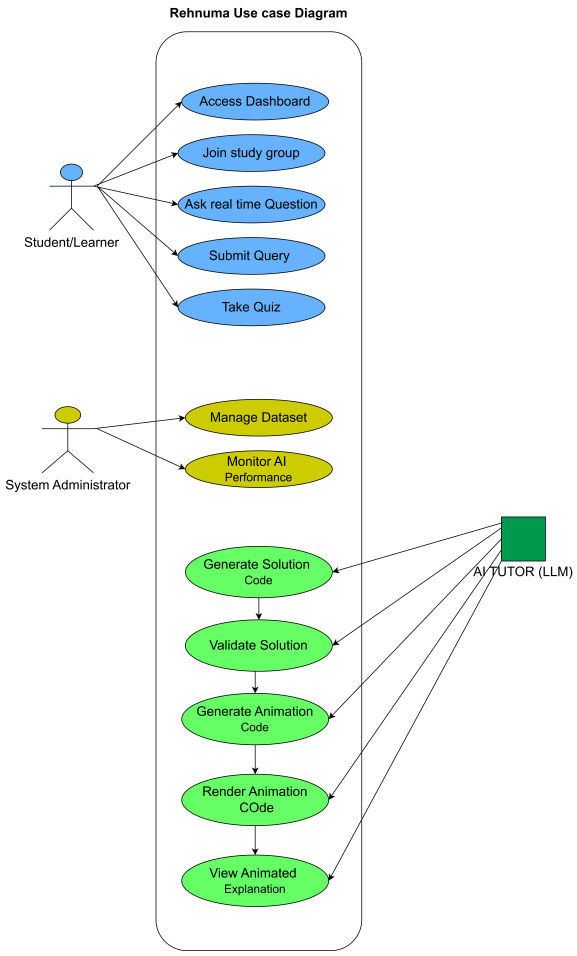

Figure A.1: Use Case Diagram for Rehnuma

A.1.2 Detail Use Case Example

Use Case ID	UC-001
Use Case Name	Generate Animated Explanation
Primary Actor	Student
Secondary Actors	Groq API (Mistral 7B, Code Llama 3.3 70B), ChromaDB, Manim Renderer
Description	Student submits a query about probability/statistics and receives an animated video explanation with audio narration using RAG pipeline.
Trigger	Student clicks "Generate Animation" button after entering query.
Preconditions	[leftmargin=*,topsep=0pt,itemsep=0pt] • Student is logged in • Valid Groq API keys configured • ChromaDB contains indexed problems
Postconditions	**Success:** Animation generated, saved, and displayed to student **Failure:** Error message displayed, query logged for review

Normal Flow

#	Actor Action	System Response
1	Student enters query: *"Explain Bayes' Theorem"*	Validates query format
2		Applies prompt engineering to enhance query
3		Searches ChromaDB for 3 similar reference problems
4		Sends enhanced query + context to Mistral 7B via Groq API
5		Receives step-by-step solution, validates with SymPy
6		Sends solution to Code Llama 3.3 70B to generate Manim code
7		Renders video (Manim) and generates audio (gTTS) in parallel
8		Synchronizes video + audio using FFmpeg
9		Saves animation to storage, updates database
10		Displays animation with play/download controls
11	Student watches animation	Streams video at 720p with audio

Alternative Flows	**A1 - Invalid Query:** System displays error, prompts correction **A2 - No Similar Problems:** Uses default examples as context **A3 - Validation Fails:** Retries up to 3 times with modified prompt **A4 - API Rate Limit:** Implements exponential backoff, waits and retries **A5 - Rendering Timeout:** Offers text-based solution as alternative
Exception Flows	**E1 - Generation Fails:** Displays error with suggestions to rephrase **E2 - API Unavailable:** Shows maintenance message, offers notification **E3 - Session Expires:** Saves animation, prompts re-login to access
Business Rules	[leftmargin=*,topsep=0pt,itemsep=0pt] • 20 animation requests per student per day • Query length: 5-500 characters • Maximum rendering time: 120 seconds • Animations stored for 90 days
Special Requirements	**Performance:** Total time ¡ 90 seconds (95% of queries) **Security?** Input sanitization, API keys in environment variables **Usability:** Loading indicators, clear error messages

Table 1: Detailed Use Case: Generate Animated Explanation

Figure A.2: Detailed Use Case Example of Rehnuma

A.2 Appendix B

A.2.1 Class Diagram

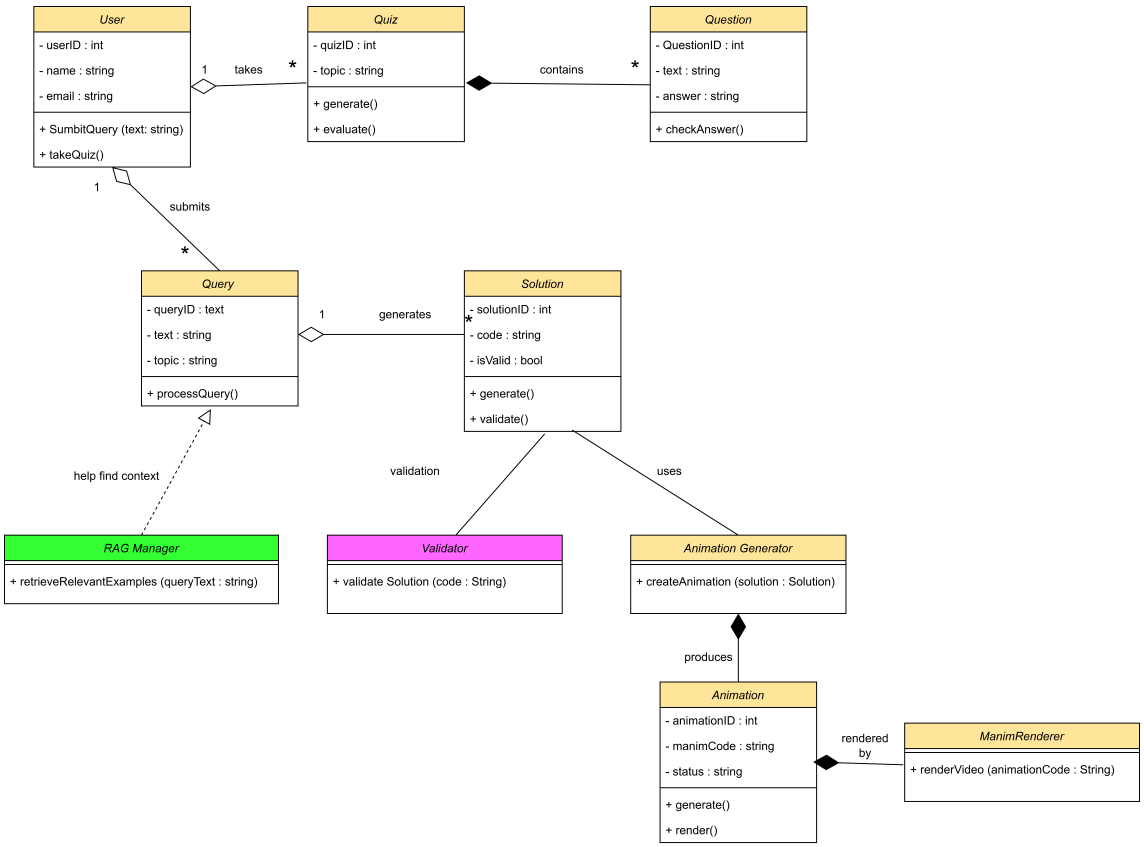

Figure A.3: Class Diagram for Rehnuma

A.3 Appendix C

A.3.1 Architecture Pattern Example

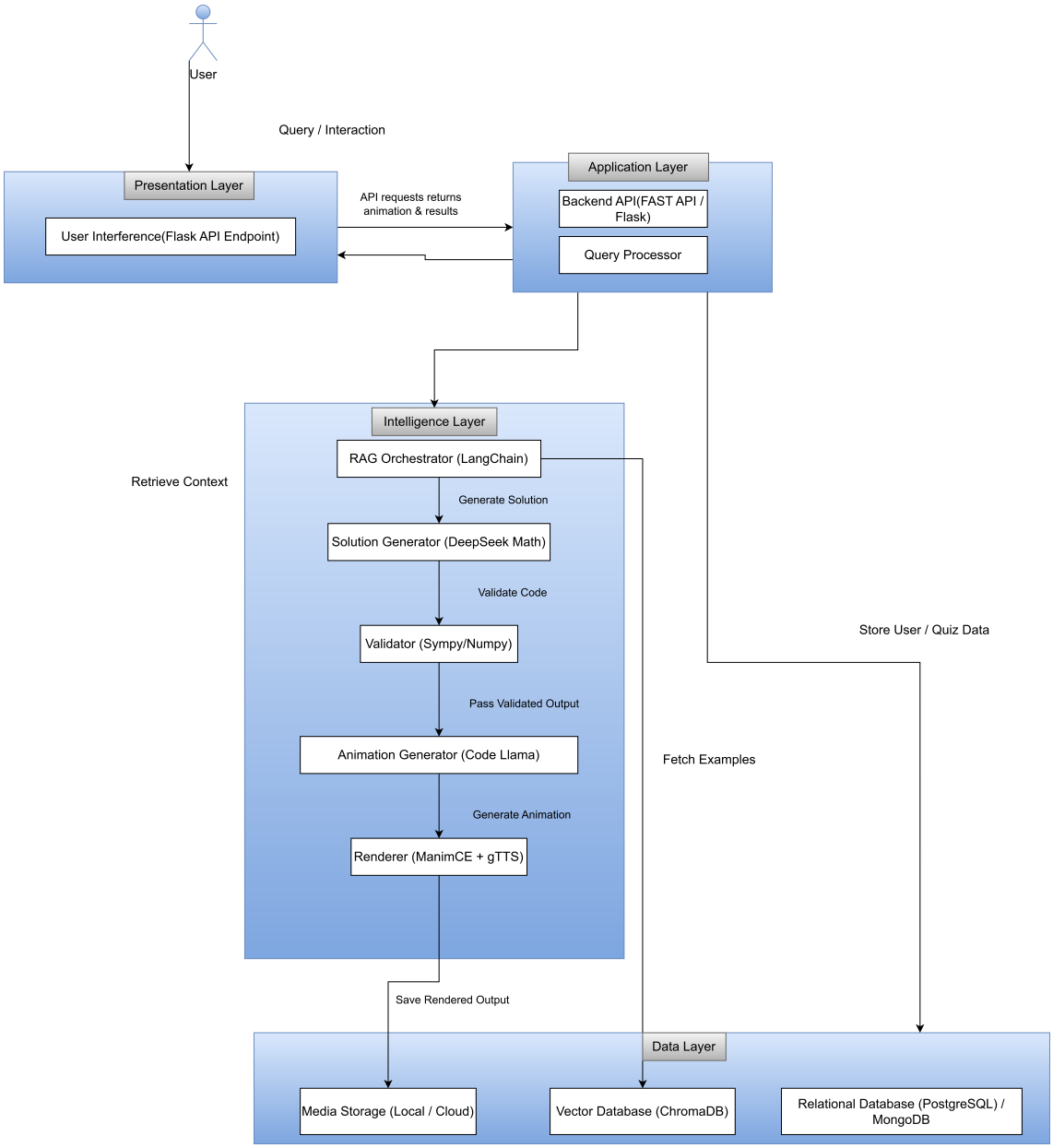

Figure A.4: Architecture Pattern For Rehnuma

A.4 Appendix D

A.4.1 Activity Diagram

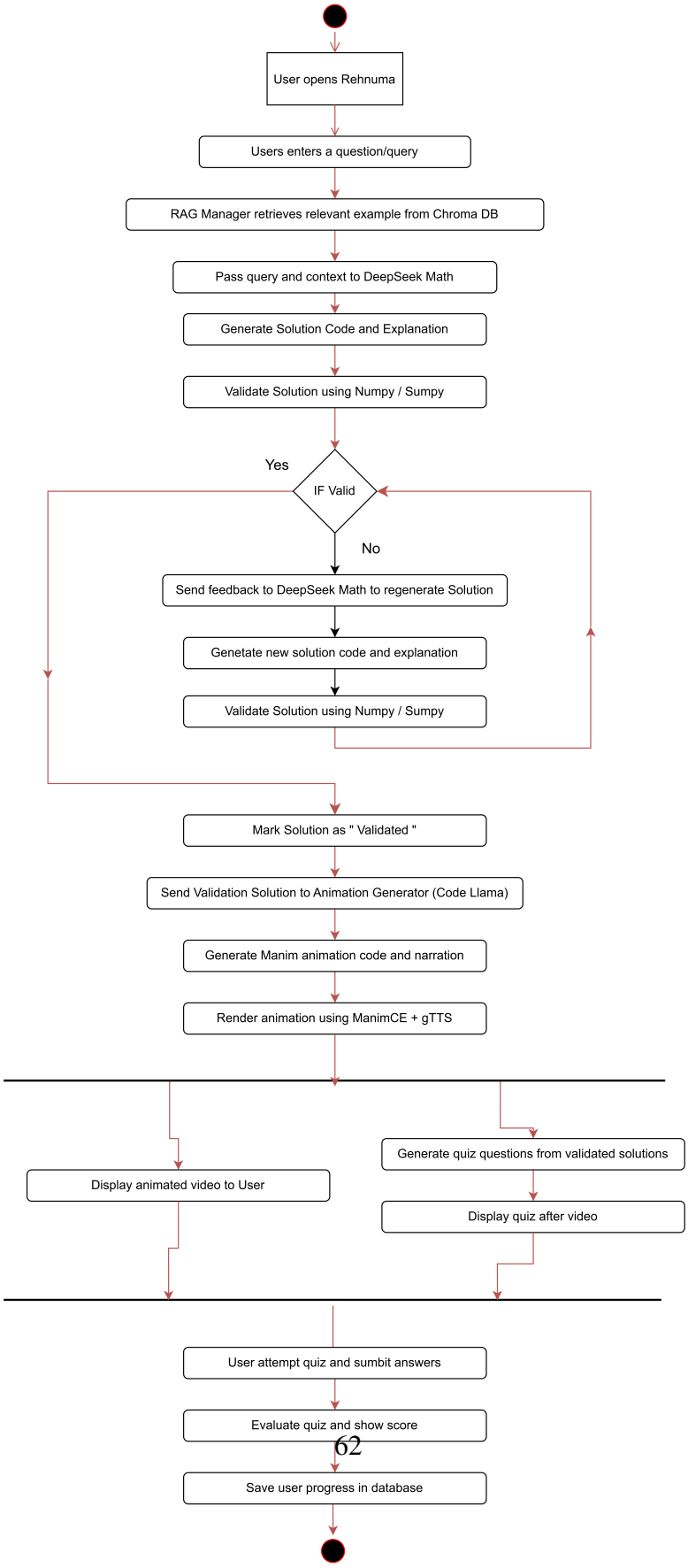

Figure A.5: Activity Diagram For Rehnuma

A.4.2 Sequence Diagram

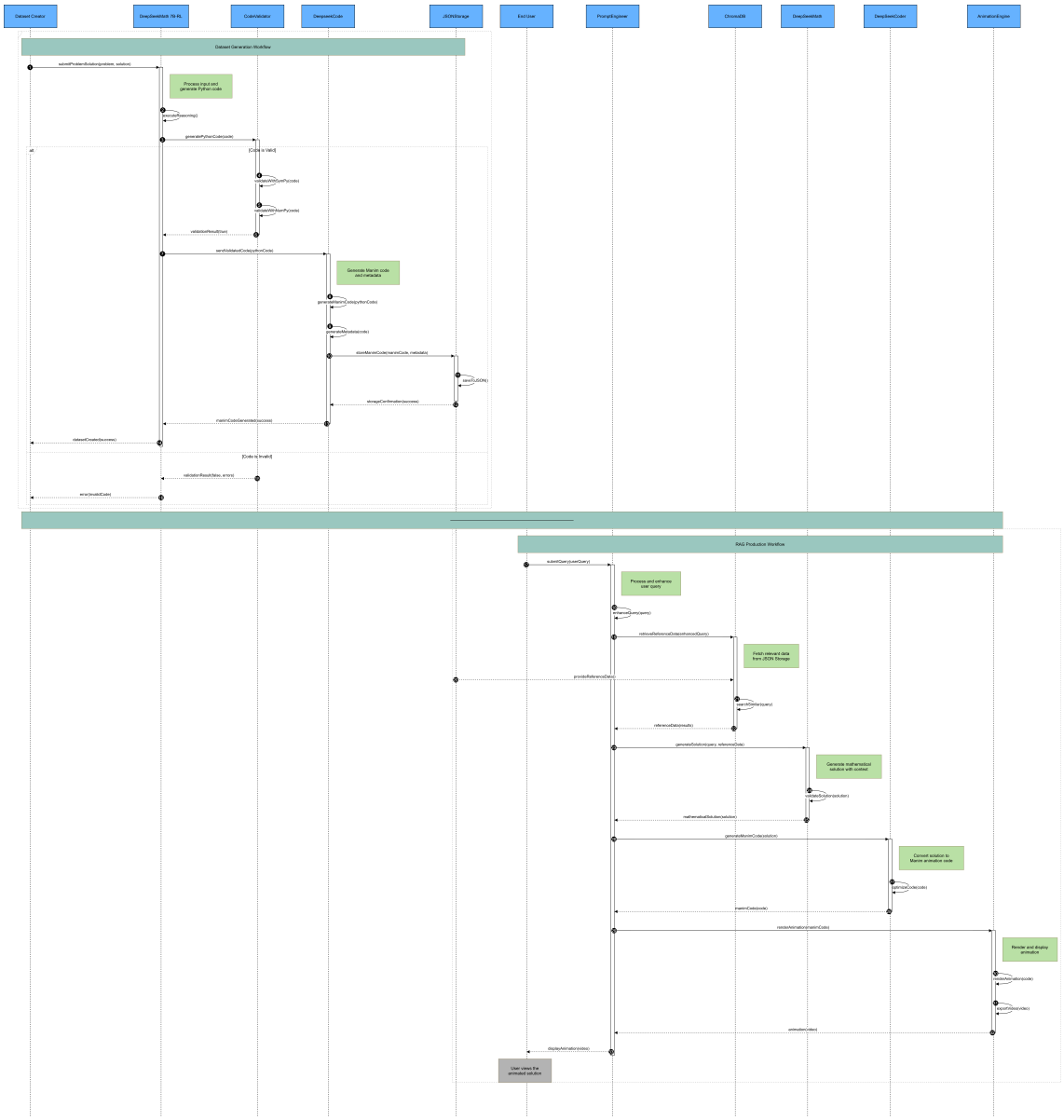

Figure A.6: Sequence Diagram

A.4.3 Data Flow Diagram

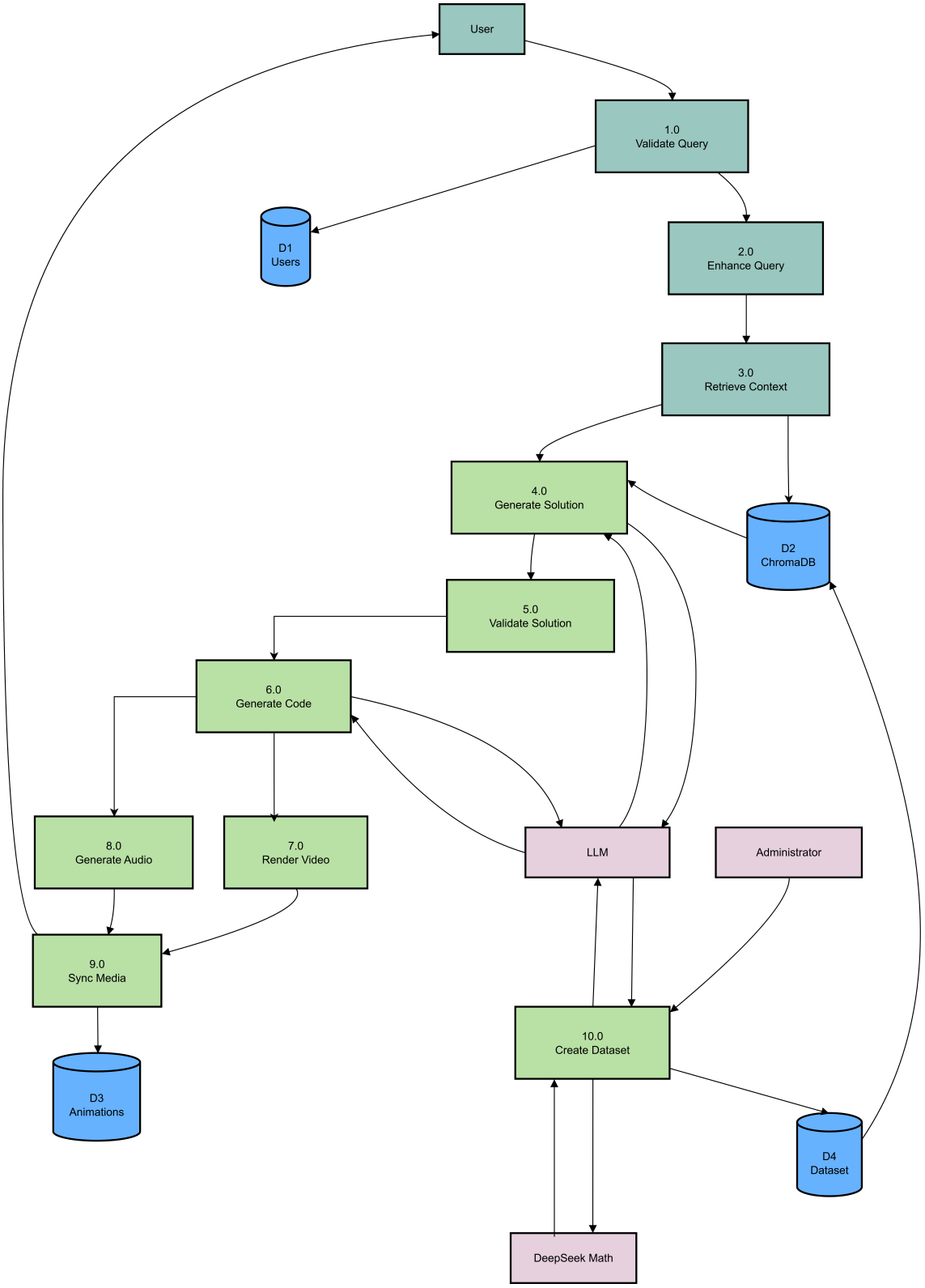

Figure A.7: Data Flow Diagram for Rehnuma