SmartStudy Companion: An AI-powered adaptive learning platform for personalized programming education, using knowledge graphs

Project Team

Laiba Batool 21i-1781
Anjuman Shaheen 21i-2664
Muhammad Ilyas 21i-2593

Session 2021-2025

Supervised by

Mr. Aqib Rehman

Co-Supervised by

Dr. Muhammad Arshad Islam

Department of Data Science

National University of Computer and Emerging Sciences
Islamabad, Pakistan

June, 2024

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document specifies the requirements for SmartStudy Companion, an AI-powered web application designed to enhance programming education through personalized learning experiences.

Intended Readers:

Project Supervisor: To evaluate the project's scope.

Student Developers (Project Team): As a guide for implementation and development.

Peer Reviewers: For providing feedback and potential collaboration.

FYP Committee: To evaluate the project's complexity and innovation.

1.1 Problem Statement

The field of programming education faces significant challenges in providing personalized, adaptive learning experiences that cater to individual student needs and learning paces. Traditional methods often struggle to address the diverse backgrounds and skill levels of students, leading to disengagement and inefficient learning processes. Moreover, the rapid evolution of programming languages and technologies makes it difficult for educational content to remain current and relevant. The SmartStudy Companion aims to address these issues by leveraging advanced AI technologies, specifically graph-based Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs).

This system is being developed to revolutionize the way programming is taught and learned, providing a solution that adapts to each student's unique learning journey. By creating personalized knowledge graphs, the SmartStudy Companion can visualize complex programming concepts as interconnected nodes, helping students understand relationships between different topics more intuitively. The system's ability to offer in-depth

explanations, tailored examples, and recommended learning paths addresses the need for individualized instruction at scale. Furthermore, the integration of a RAG-powered chatbot enables students to get immediate, context-aware answers to their programming queries, simulating a one-on-one tutoring experience.

The SmartStudy Companion not only aims to enhance the effectiveness of programming education but also seeks to make it more engaging and accessible. By providing clear progressions from fundamental to advanced topics and illustrating the practical applications of programming concepts, the system aims to maintain student motivation and interest. Ultimately, this project seeks to bridge the gap between traditional teaching methods and the dynamic, personalized learning experiences required in today's fast-paced technological landscape, potentially setting a new standard for intelligent educational tools in computer science education.

1.2 Scope

The SmartStudy Companion project encompasses the development of an intelligent, adaptive learning system focused on programming concept education. At its core, the system will utilize a graph-based Retrieval-Augmented Generation (RAG) pipeline and Large Language Models (LLMs) to create personalized learning experiences. The primary scope includes the creation of a user interface that allows learners to interact with the system, view their personalized knowledge graphs, and engage with the AI-powered chatbot for programming-related conceptual queries. The system will initially focus on a specific set of fundamental programming concepts, with the potential for expansion in future iterations.

Key functionalities within the scope include:

- User registration and profile creation to store individual learning progress and conceptual understanding.
- Implementation of the RAG pipeline to process and structure programming concept-related content from predefined sources.
- Generation of personalized knowledge graphs for each user, visually representing their current understanding of programming concepts and learning progress.
- Development of an adaptive assessment system that adjusts difficulty based on user performance and updates the knowledge graph accordingly.
- Integration of a chatbot interface powered by LLMs to provide immediate, context-aware explanations for programming concepts.

- Creation of a recommendation engine that suggests personalized learning paths based on the user's conceptual knowledge graph and learning goals.
- Implementation of interactive concept exploration tools that allow users to visualize and interact with programming concepts directly within the platform.
- Development of a progress tracking system that provides users with insights into their conceptual understanding and learning journey.

The project will also include the design and implementation of a backend system to manage user data, store and process the conceptual knowledge graphs, and handle the integration between various components.

1.3 Modules

1.3.1 Data collection and preprocessing

This module focuses on gathering and preparing the necessary data for the SmartStudy Companion system.

- Collect programming books data
- Clean that data
- Perform chunking
- Handle duplicate data

1.3.2 Model building (Knowledge graph)

This module involves creating a structured knowledge representation of programming concepts.

- Select or train model for knowledge extraction
- Create knowledge graph structure
  - Entity recognition
  - Relationship recognition
- Make knowledge graph in hierarchy to show concepts from basic to advanced
- Store knowledge graph data in database

1.3.3 Model building (RAG)

This module focuses on implementing Retrieval-Augmented Generation for efficient information retrieval.

- Generate graph embedding for efficient retrieval for implementing RAG
- Implement similarity search on graph embeddings
- Store in database

1.3.4 Information retrieval

This module handles the retrieval and presentation of relevant information to users.

- Fetch relevant subgraphs based on user expertise or skills
  - Visualize the graph
- Show details of concepts when user click on specific node
- Develop a Chat Bot using RAG (to answer queries of users related to programming)

1.3.5 Quiz and Assessment

This module manages user evaluation and progress tracking.

- Implement adaptive testing based on user performance
- Design a system to track user progress across topics
- Suggest relevant topics according to user performance

1.3.6 User interface

This module focuses on creating an engaging and user-friendly interface for the system.

- Design and implement an intuitive, user-friendly interface
- Implement user login and store login information in database
- Create interactive visualizations for knowledge graphs
- Design interface for chat interactions with the RAG-powered chatbot

1.3.7 Testing

This module ensures the quality and reliability of the SmartStudy Companion system.

- Fix bugs
- Perform unit testing to evaluate model
- Perform Integration testing
- Gather feedback from users to improve system performance

1.4 User Classes and Characteristics

User class	Description
Student	A Student is the primary user of the SmartStudy Companion, typically a fellow university student looking to understand programming concepts. For this FYP, we anticipate testing with 20-30 students from our university. Students will access the system to explore concepts, take assessments, interact with the AI chatbot, and track their learning progress. They will primarily access the system through web browsers on desktop computers or laptops.
Educator	Educators are primarily the project supervisors and potentially 1-2 other faculty members who will evaluate and provide feedback on the SmartStudy Companion. They will use it to assess the system's effectiveness in teaching programming concepts and may suggest improvements or additional features. Educators will access the system through web browsers on desktop computers.
Administrator	The Administrator role will be shared among the three team members developing the FYP. As Administrators, we will have full access to all system features for development, testing, and demonstration purposes. This includes managing the concept database, monitoring system performance, and making necessary adjustments. We will access the system through both regular user interfaces and development interfaces.

Chapter 2

Project Requirements

2.1 Use-case/Event Response Table/Storyboarding

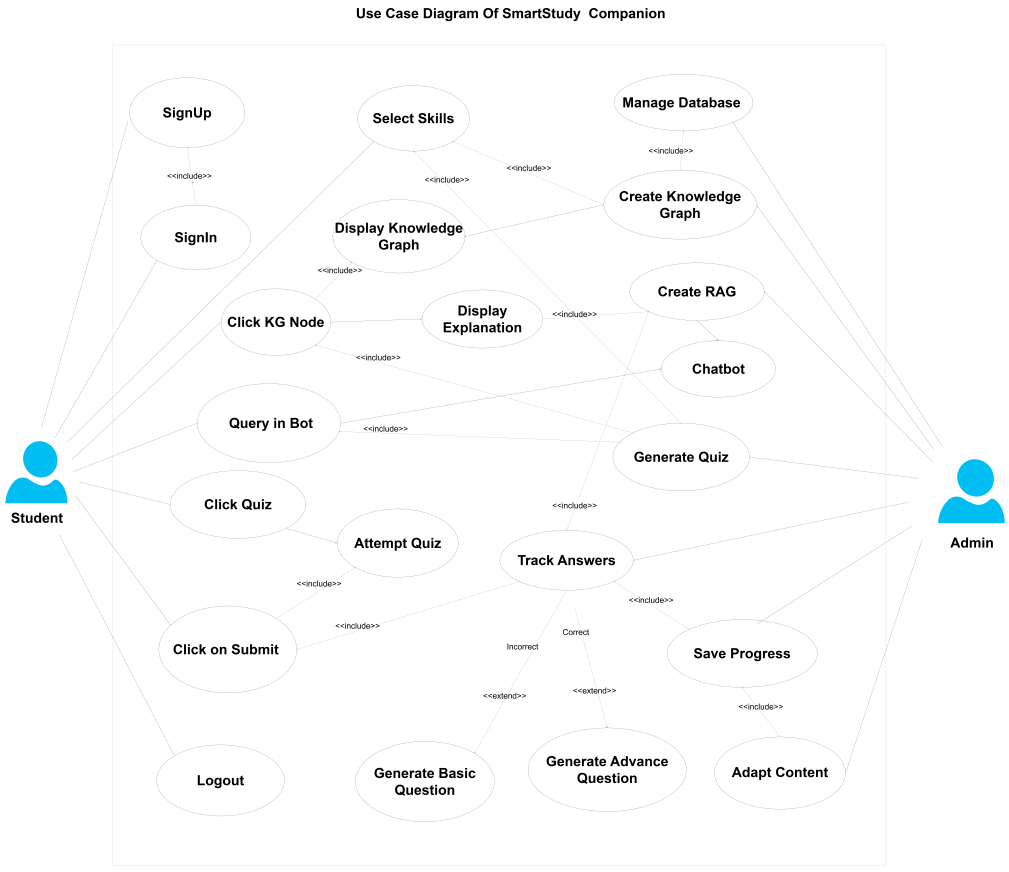


Figure 2.1: Use case SmartStudy Companion

2.2 Functional Requirements

2.2.1 Data Collection and Preprocessing Module

Following are the requirements for the Data Collection and Preprocessing Module:

FR1: Collect programming books.

FR2: The system shall clean the collected data by removing irrelevant information and formatting inconsistencies.

FR3: The system shall perform text chunking on the cleaned data to create manageable segments.

FR4: The system shall identify and remove duplicate data to ensure data integrity.

2.2.2 Model Building (Knowledge Graph) Module

Following are the requirements for the Model Building (Knowledge Graph) Module:

FR5: The system shall select or train a model capable of extracting knowledge from preprocessed text data.

FR6: The system shall create a knowledge graph structure by identifying entities and relationships within the text.

FR7: The system shall organize the knowledge graph in a hierarchical manner, representing concepts from basic to advanced.

FR8: The system shall store the created knowledge graph data in a database.

2.2.3 Model Building (RAG) Module

Following are the requirements for the Model Building (RAG) Module:

FR9: The system shall generate graph embeddings for efficient retrieval in the RAG implementation.

FR10: The system shall implement a similarity search function on the graph embeddings.

FR11: The system shall store the generated embeddings in a database.

2.2.4 Information Retrieval Module

Following are the requirements for the Information Retrieval Module:

FR12: The system shall fetch relevant subgraphs based on user expertise or skills.

FR13: The system shall provide a visualization of the retrieved subgraphs.

FR14: The system shall display detailed information about concepts when a user selects a specific node.

FR15: The system shall implement a chatbot using RAG to answer user queries related to programming.

2.2.5 Quiz and Assessment Module

Following are the requirements for the Quiz and Assessment Module:

FR16: The system shall provide a quiz after the completion of each learning topic.

FR17: The system shall implement adaptive testing that adjusts question difficulty based on the user's previous answer.

FR18: The system shall present quiz questions one at a time.

FR19: The system shall track the number of consecutive incorrect answers during a quiz.

FR20: The system shall terminate the quiz if the user provides a specified number of consecutive incorrect answers.

FR21: The system shall recommend that the user revisit the topic if the quiz is terminated due to consecutive incorrect answers.

FR22: The system shall store quiz results and use them to update the user's knowledge graph.

FR23: The system shall use quiz performance to suggest relevant topics for further study.

2.2.6 User Interface Module

Following are the requirements for the User Interface Module:

FR24: The system shall provide an intuitive and user-friendly interface for all functionalities.

FR25: The system shall implement a user login system and store login information in a secure database.

FR26: The system shall create interactive visualizations for displaying knowledge graphs.

FR27: The system shall design and implement an interface for chat interactions with the RAG-powered chatbot.

FR28: The system shall provide a dedicated interface for quizzes, including:

(a) A section to display one quiz question at a time.

(b) A mechanism for users to input and submit their answer to each question.

(c) A 'Next' button to proceed to the subsequent question after answering.

(d) Immediate feedback on the correctness of each answer.

(e) A quiz termination screen that appears when the user provides too many consecutive incorrect answers.

FR29: The system shall display a recommendation to revisit the topic if a quiz is terminated due to poor performance.

FR30: The system shall integrate seamlessly between learning interfaces and quiz interfaces to provide a smooth user experience.

2.2.7 Testing Module

Following are the requirements for the Testing Module:

FR31: The system shall include a mechanism for identifying and fixing bugs.

FR32: The system shall perform unit testing to evaluate individual components of the model.

FR33: The system shall conduct integration testing to ensure proper functioning of all modules together.

FR34: The system shall provide a mechanism for gathering and incorporating user feedback to improve system performance.

2.3 Non-Functional Requirements

2.3.1 Reliability

For our SmartStudy Companion project, reliability is crucial. We need to make sure our system works consistently for students and doesn't disrupt their learning. Here are our key reliability requirements:

1. Our system should work without any major issues for at least 14 days straight. This means no crashes or data losses during this period.

2. We define a major failure as:

    - Users unable to log in for more than 5 minutes
    - Knowledge graph not loading or updating properly
    - Chatbot not responding to user queries for more than 2 minutes

3. If our system fails, students might lose their quiz progress or be unable to access study materials. This could frustrate users and impact their learning, so we need to minimize these occurrences.

4. To protect from failures, we will:

    - Automatically save quiz answers every minute
    - Keep a backup of the knowledge graph that updates daily

5. For error detection, we'll set up monitoring that alerts us if:

    - Any page takes more than 60 seconds to load
    - The chatbot fails to respond to more than 5 consecutive queries
    - More than 10 users report the same issue within an hour

6. To fix errors quickly, we'll:

    - Keep a recent backup of user data to restore in case of data corruption

These reliability measures will help ensure our SmartStudy Companion provides a stable and trustworthy learning experience for students.

2.3.2 Usability

Following usability requirements aim to create an intuitive, efficient, and user-friendly learning experience for all users of the SmartStudy Companion.

USE-1: The system shall display an explanation of a concept on the side panel when a user clicks on a node in the knowledge graph.

USE-2: The knowledge graph visualization shall provide zoom capabilities, allowing users to navigate through concepts with intuitive mouse interactions.

USE-3: The system shall use color-coding for nodes in the knowledge graph to indicate the user's familiarity with concepts:

- Red nodes shall represent concepts the user does not know.
- Green nodes shall represent concepts the user knows.

USE-4: The system shall provide clear error messages for incorrect quiz answers.

USE-5: The web interface shall be responsive, ensuring proper display and functionality across different desktop browser window sizes.

USE-6: The quiz interface shall allow users to easily navigate between questions and submit their answers with minimal clicks.

USE-7: The chatbot interface shall be easily accessible from any page of the web application, providing quick access to assistance.

2.3.3 Performance

The SmartStudy Companion shall meet the following performance requirements:

PER-1: The system shall retrieve and visualize relevant subgraphs within 10 seconds of a user request.

PER-2: The chatbot shall provide responses to user queries within 10 seconds for 90% of queries.

PER-3: Quiz questions shall be generated and displayed within 10 second of the previous question being answered.

PER-4: The system shall update the user's knowledge graph within 30 seconds of completing a quiz.

PER-5: All database operations, including storing and retrieving user data, shall complete within 20 seconds.

2.3.4 Security

The system shall ensure that each user can only access their own learning progress, quiz results, and personal information.

The system shall implement access controls to prevent unauthorized users from viewing or modifying the knowledge graph and learning content.

2.4 Domain Model

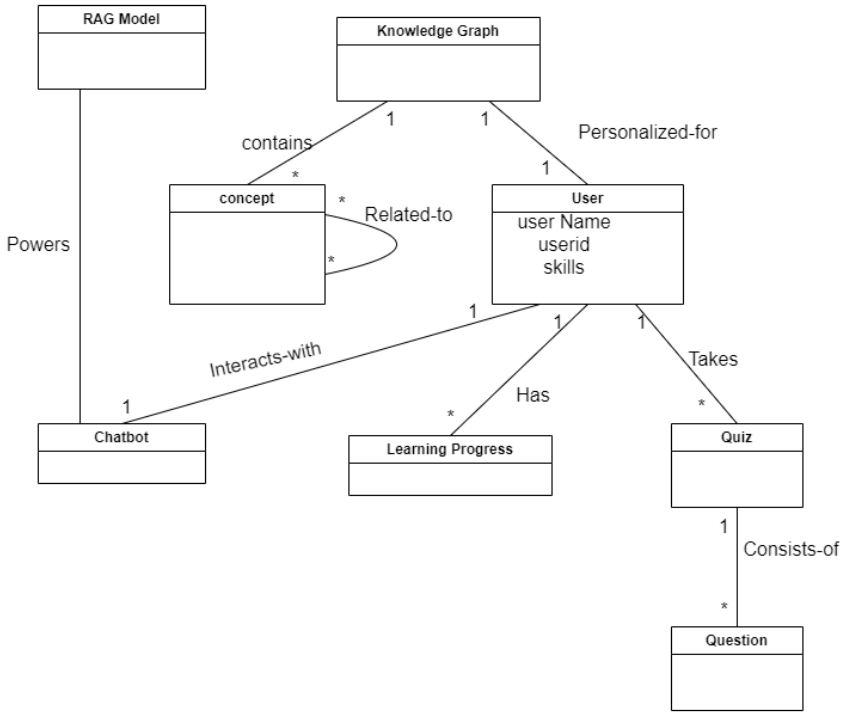
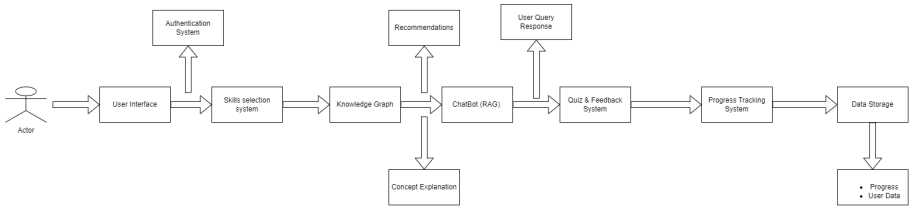


Figure 2.2: Domain Model for SmartStudy Companion

Chapter 3

System Overview

3.1 Architectural Design



Architechture_Diagram

Figure 3.1: Architecture Diagram

3.2 Design Models

3.2.1 Activity Diagram

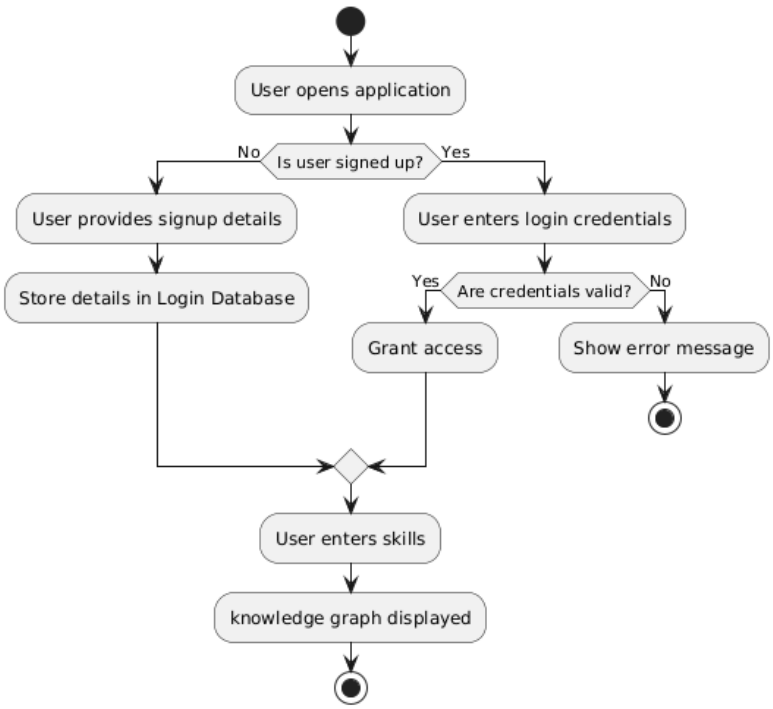
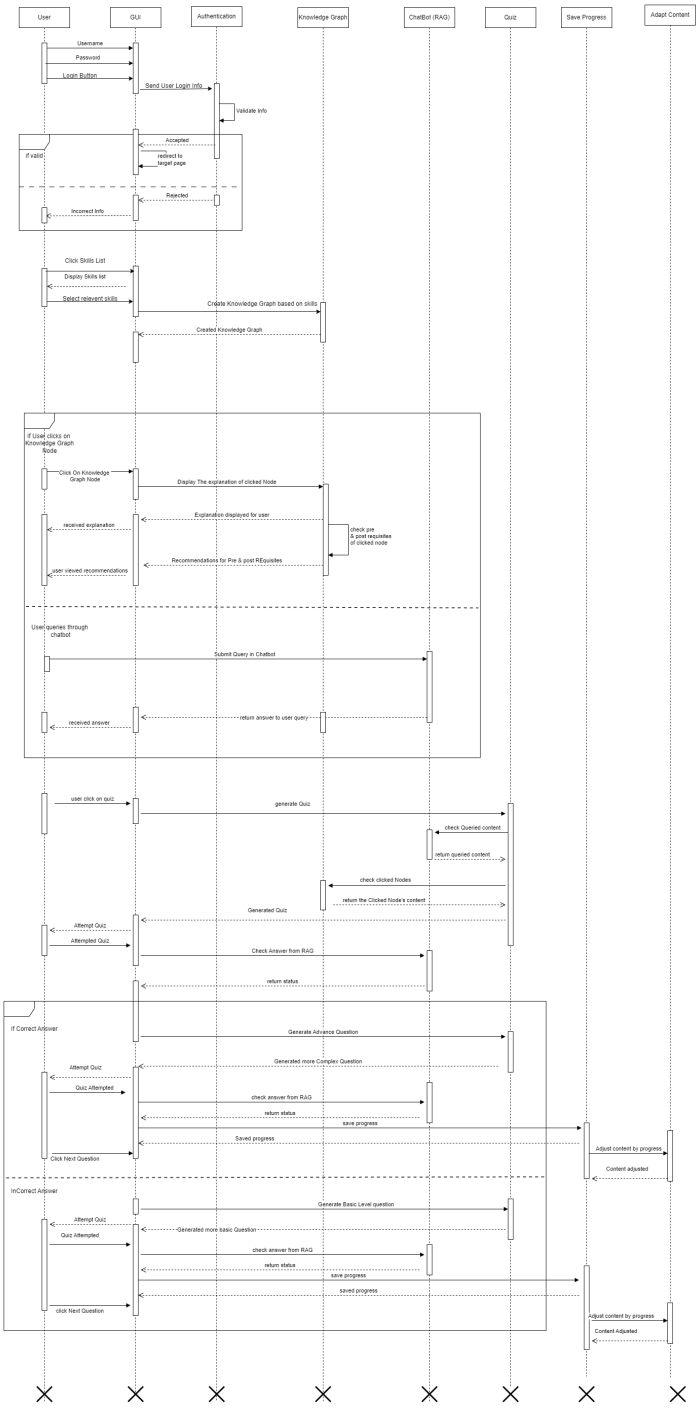


figure Activity Diagram

3.2.2 System Sequence Diagram



Sequence Diagram

figure sequence diagram

3.2.3 state Transition Diagram

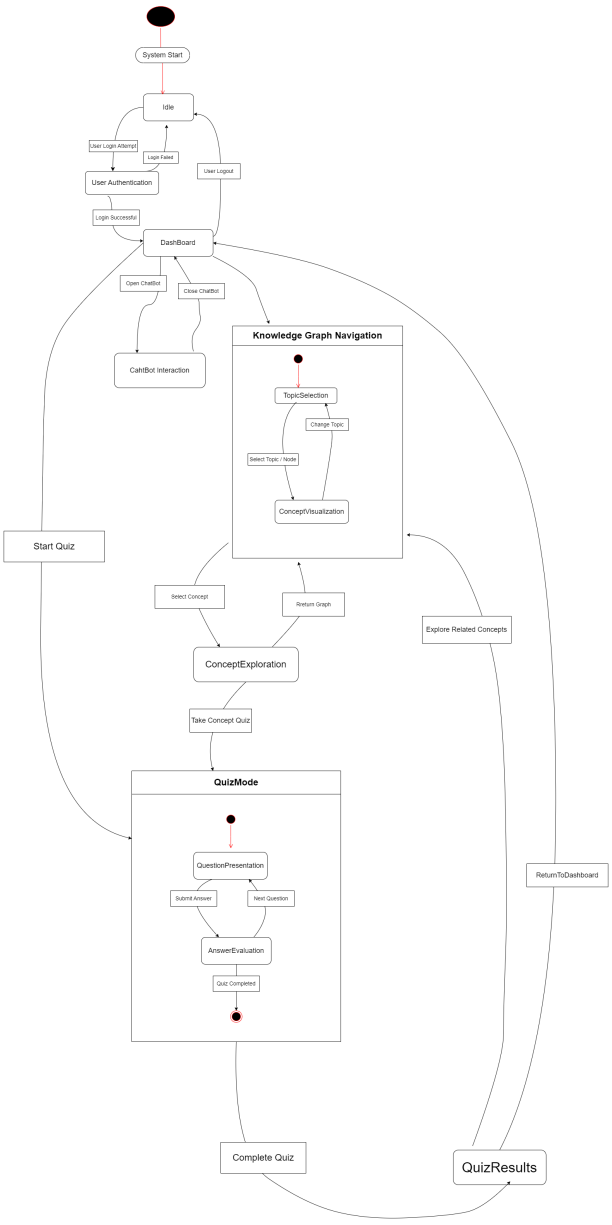


Figure 3.2: state Transition Diagram

3.2.4 Data Flow Diagram

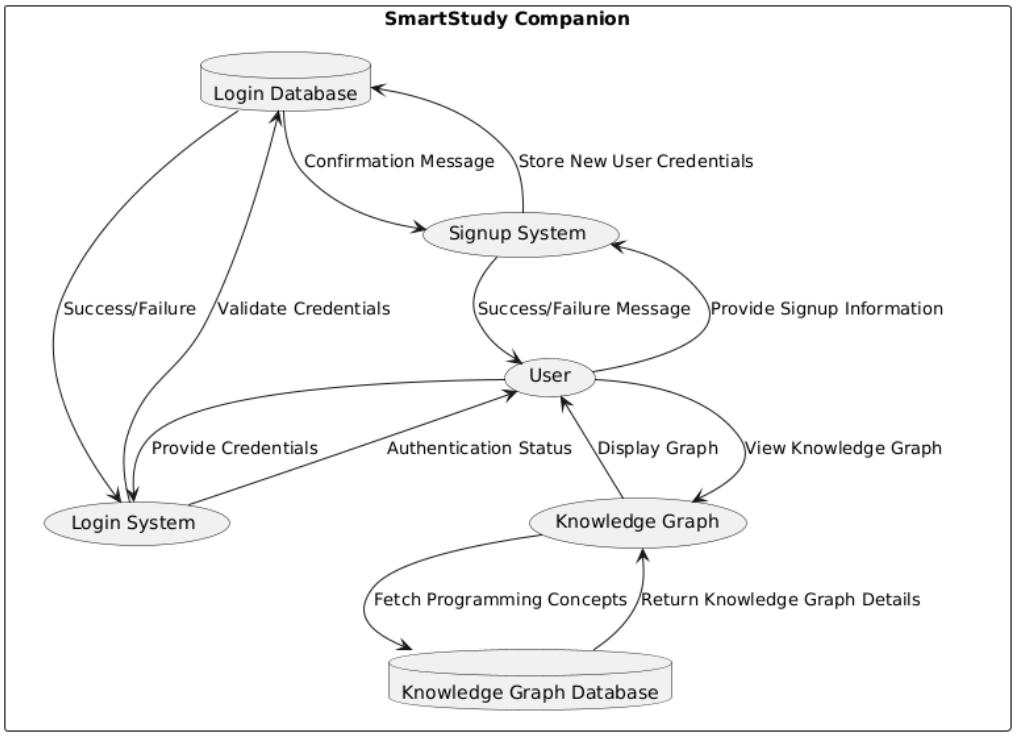


Figure 3.3: Data Flow Diagram

3.3 Data Design

3.3.1 Data Structures and Storage

The system primarily uses two databases:

- User data is stored in MongoDB, a document-based NoSQL database.
- The Knowledge Graph is stored in Neo4j/ GraphDB, a graph database.

3.3.2 Data Organization

- User documents contain personal information and references to skills.
- The Knowledge Graph represents concepts and their relationships.

3.3.3 JSON Schemas

The following JSON schemas define the structure for user data and knowledge graph nodes, ensuring data integrity and consistency across the system.

3.3.3.1 User Data Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "userId": { "type": "string", "format": "uuid" },
    "email": { "type": "string", "format": "email" },
    "name": { "type": "string" },
    "password": { "type": "string", "minLength": 8 },
    "skills": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "conceptId": { "type": "string" },
          "level": { "type": "integer", "minimum": 1, "maximum": 5 }
        },
        "required": ["conceptId", "level"]
      }
    }
  },
  "required": ["userId", "email", "name", "password"]
}
```

3.3.3.2 Knowledge Graph Node Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "conceptId": { "type": "string" },
    "name": { "type": "string" },
    "description": { "type": "string" },
    "relationships": {
```

```
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "relatedConceptId": { "type": "string" },
          "relationType": { "type": "string", "enum": ["prerequisite", "related", "
        },
        "required": ["relatedConceptId", "relationType"]
      }
    }
  },
  "required": ["conceptId", "name", "description"]
}
```

These schemas define the structure for user data and knowledge graph nodes, ensuring data integrity and consistency across the system. The user data schema includes fields for personal information and skills, while the knowledge graph node schema represents individual concepts and their relationships within the learning domain.

Chapter 4

Implementation and Testing

4.1 System Overview

The *SmartStudy Companion* is a personalized learning platform designed specifically for programming students and enthusiasts. It integrates a hierarchical knowledge graph, an adaptive quiz engine, and a RAG-based (Retrieval-Augmented Generation) chatbot to provide a tailored and interactive learning experience. The system is built to dynamically assess a user's knowledge, guide them through appropriate learning paths, and support their understanding through intelligent query resolution.

Functionality Overview: Upon registration, users select programming topics they believe they've mastered. The system verifies this through a multi-level quiz (basic, medium, advanced). Based on quiz performance, users are shown either prerequisite or more advanced topics via a knowledge graph powered by Neo4j. As users interact with the graph or chatbot, their learning progress is continuously updated.

The quiz module uses the Code LLaMA model to generate context-appropriate multiple-choice questions. A RAG-based chatbot assists users by answering programming-related queries using a curated document base, ensuring relevant and up-to-date responses.

Design and Technology Stack: The frontend is built with Streamlit for a user-friendly web interface. The backend, developed in Python, manages user tracking, quiz logic, graph traversal, and chatbot integration. Neo4j serves as the graph database, and Ngrok was used during development to expose the local app for testing.

Testing: Both unit and integration tests were conducted to validate functionality and ensure system reliability. Unit testing was implemented using `unittest` and `pytest`, covering quiz generation, graph updates, chatbot accuracy, and UI components. Integration testing confirmed that modules worked cohesively, preserving consistent user flow and data integrity.

4.2 Algorithm Design

This section describes the core algorithms developed to implement the major functionalities of the *SmartStudy Companion* system. The system comprises three main modules: the Knowledge Graph Module, the Quiz Module, and the RAG-based Chatbot Module. Each algorithm is designed to ensure personalized learning progression, accurate skill assessment, and interactive support for programming concepts. The following diagrams illustrate the logic and structure of the core algorithms used in each module. .

InitialKnowledgeAssessment: This algorithm verifies whether the user truly understands the programming topics they have marked as already known. It generates a multi-level quiz for each selected topic and evaluates user responses. Based on the score, the system determines whether the user can advance to the next level or must revisit foundational concepts.

```
Algorithm 1 InitialKnowledgeAssessment
Input: Set of user-selected known topics T_known, QuestionBank QB
Output: Verified topics T_verified, Updated knowledge level K_level

T_verified ← ∅
score ← 0
For each topic t in T_known do
    Q_set ← generateQuiz(QB, t, levels = {basic, medium, advanced})
    ans ← getUserResponses(Q_set)
    result ← evaluate(ans)
    If result.score ≥ threshold then
        T_verified ← T_verified ∪ {t}
    end if
end for
If |T_verified| ≥ |T_known| * 0.7 then
    K_level ← nextLevel(T_verified)
Else
    K_level ← basicLevel()
End if
```

Figure 4.1: InitialKnowledgeAssessment Algorithm

KnowledgeGraphNavigator: This algorithm determines which nodes of the knowledge graph to display, based on the user's most recent quiz performance. If the user passes, advanced nodes are displayed; otherwise, prerequisite/basic nodes are shown. This ensures personalized and progressive learning.

```
Algorithm 2 KnowledgeGraphNavigator
Input: UserKnowledgeState U_state, KnowledgeGraph KG
Output: Nodes to display N_display

1: N_display ← ∅
2: current_level ← U_state.currentLevel
3: If U_state.lastQuizPassed = true then
4:     N_display ← KG.getNextLevelNodes(current_level)
5: Else
6:     N_display ← KG.getPrerequisiteNodes(current_level)
7: End if
8: Display N_display to user
```

Figure 4.2: KnowledgeGraphNavigator Algorithm

AdaptiveQuizGenerator: This algorithm dynamically generates a personalized quiz using topics the user has studied via chatbot queries or knowledge graph interactions. It selects questions from each difficulty level—basic, medium, and advanced—ensuring a balanced and accurate assessment.

```
Algorithm 3 AdaptiveQuizGenerator
Input: Set of topics T_learned, QuestionBank QB
Output: Set of MCQ quiz questions Q_set

1: Q_set ← ∅
2: For each topic t in T_learned do
3:     For level in {basic, medium, advanced} do
4:         q ← selectRandom(QB[t][level])
5:         Q_set ← Q_set ∪ {q}
6:     End for
7: End for
8: Return Q_set
```

Figure 4.3: AdaptiveQuizGenerator Algorithm

4.3 External APIs/SDKs

The implementation of the *SmartStudy Companion* system incorporates several external APIs and SDKs to support its core functionalities, including intelligent quiz generation, interactive chatbot responses, frontend interface rendering, and structured knowledge storage. These tools were essential for enabling scalable AI-based interactions, building a responsive user interface, and managing hierarchical programming concepts efficiently. The following table summarizes the key APIs/SDKs used in the project along with their specific roles and functions.

API/SDK and Version	Description	Purpose of Usage	API Endpoint/Function/Class Used
Code LLaMA (Meta, 7B/13B)	Large Language Model for code tasks	Used for generating adaptive quizzes and answering programming queries through RAG-based chatbot	llama.generate(prompt), llama.chat()
Neo4j (v5)	Graph database management system	Used to store and query the hierarchical knowledge graph of programming topics	MATCH, CREATE, Cypher queries via neo4j-python driver
Streamlit	Python-based web app framework	Frontend interface for interactive learning, quizzes, and chatbot access	streamlit.write(), streamlit.button(), streamlit.chat*input*()
Ngrok	Secure tunneling service	Used to expose the locally hosted Streamlit app to the internet for testing and access	ngrok http 8501

4.4 Testing Details

Once the *SmartStudy Companion* system was successfully developed, thorough testing was conducted to ensure that each component operated as intended. Testing aimed to verify that the system met its defined functional requirements, such as adaptive learning progression, quiz accuracy, knowledge graph responsiveness, and chatbot reliability. Additionally, testing helped uncover hidden errors and edge case failures, enabling early fixes before deployment. A combination of unit testing, integration testing, and manual validation was used during development. This ensured that the system was stable, met user expectations, and delivered a seamless learning experience.

4.4.1 Unit Testing

Unit testing was applied across all core modules of the system to verify the correctness of individual functionalities in isolation. The main focus was on three components:

- **Knowledge Graph Module:** Tested functions for fetching related topic nodes, traversing dependencies, and updating user progress level in the Neo4j graph database.
- **Quiz Generation Module:** Verified the accuracy of question generation using the Code LLaMA model, including question filtering by difficulty level (basic, medium, advanced) and ensuring proper formatting of multiple-choice answers.
- **Chatbot Module:** Ensured that the RAG-based chatbot returned relevant and context-aware answers to programming queries and did not respond to irrelevant or unsupported topics.

Table 4.1: Unit Testing Summary for SmartStudy Companion

Module	**Test Case**	**Description**	**Result**
Knowledge Graph Module	Node Retrieval Accuracy	Ensure that only relevant topic nodes based on user level and quiz result are fetched from Neo4j	Pass
Knowledge Graph Module	Prerequisite Expansion	Verify that prerequisite nodes are correctly displayed when user fails quiz	Pass
Knowledge Graph Module	Node Interaction	Check that clicking a node correctly triggers the explanation view for that specific concept	Pass
Quiz Generation Module	Topic Scope Validation	Ensure that quiz questions are generated only from the selected topic and its prerequisites (e.g., selecting "Array" should not include "Pointers")	Pass
Quiz Generation Module	Difficulty-Level Accuracy	Validate that questions are correctly categorized and fetched for basic, medium, and advanced levels	Pass
Quiz Generation Module	Question Format Verification	Check that each MCQ has a proper format (one correct option, three distractors, and valid syntax)	Pass
Quiz Generation Module	Adaptive Update Logic	Confirm that the system updates user level based on quiz performance (promote/demote in knowledge graph)	Pass
RAG-based Chatbot Module	Relevant Response Testing	Ensure that chatbot returns relevant answers only to programming-related queries	Pass
RAG-based Chatbot Module	Out-of-Scope Query Handling	Validate that chatbot does not respond to unrelated/non-programming queries	Pass
RAG-based Chatbot Module	Contextual Continuity	Check that the chatbot can maintain context during a series of follow-up programming questions	Pass
Frontend (Streamlit)	UI Component Rendering	Ensure that all components (quiz, graph, chatbot input/output) load and render correctly	Pass
Frontend (Streamlit)	User Flow Integration	Verify the flow from topic selection $\rightarrow$ quiz $\rightarrow$ graph update $\rightarrow$ chatbot interaction works smoothly	Pass
System Integration	Data Flow Consistency	Validate that data between modules (e.g., quiz result $\rightarrow$ knowledge graph update) flows accurately and without delay	Pass

Chapter 5

Conclusions and Future Work

5.1 Conclusion

SmartStudy Companion, an AI-powered web application that makes learning programming easier and more personal. Our system uses knowledge graphs, large language models, and adaptive quizzes to help students learn at their own pace.

The system solves key problems in programming education by showing concepts as connected nodes in a graph. This helps students see how different topics relate to each other. Our three main algorithms—for chatbot, Knowledge graph, and quiz generation—create a truly personalized learning experience.

We built the system with a Streamlit frontend and Python backend using Neo4j as our graph database. The Code LLaMA model powers our quizzes and chatbot, giving students helpful answers to their questions, much like having a personal tutor.

Our testing showed that SmartStudy Companion works well. It can assess what a student knows, suggest appropriate topics to learn next, and answer questions accurately.

In summary, SmartStudy Companion bridges the gap between traditional teaching and modern personalized learning. By adapting to each student's needs and providing immediate help, it can improve how programming is taught and learned.

5.2 Future Work

While SmartStudy Companion is already useful, we see several ways to improve it:

5.2.1 Better Knowledge Graphs

- Add new programming concepts automatically as they emerge
- Show connections between similar concepts in different programming languages
- Let teachers and advanced users add to the knowledge graph

5.2.2 More Content and Features

- Add a code editor where users can practice directly
- Suggest coding projects based on what the user knows
- Connect with school learning systems

5.2.3 Technical Upgrades

- Create a mobile app version
- Make the system faster for more users
- Add offline features for when internet is unavailable

These improvements would make SmartStudy Companion even more useful for teaching programming and could serve as a model for AI-based learning in other subjects.

Bibliography