Software Design

Version 1

February 27th, 2026

CLASS (Cybersecurity Learning with AI for Static Systems)

Team Members: Sean Golez, William Barnett, Kayden Vicenti, Colton Leighton

Team Mentor: Scott LaRocca

Sponsor: Dr. Lan Zhang

School of Informatics, Computing, and Cyber Systems, Northern Arizona University

Table of Contents

Introduction

CLASS is a web-based AI tutoring platform designed to support students who are learning about cybersecurity through the use of static analysis capture the flag challenges (CTF). Static analysis CTFs are widely used to teach secure software design, vulnerability discovery, and program reasoning. However, these challenges are difficult for learners because they require an understanding of control flow, data flow, and program semantics without executing the program. In instructional settings, students depend on instructor and TA assistance, and the use of general-purpose AI tools. These tools provide inconsistent and inaccurate guidance due to the lack of grounded, authoritative content. Sponsored by Dr. Lan Zhang of Northern Arizona University and developed in support of the SE450 course, CLASS addresses this gap by providing scalable, structured, and conceptually grounded tutoring for static-analysis CTF challenges.

Our system will enable students to better understand the broader context and structure of CTF challenges. It will also allow instructors to categorize, visualize, and locate related challenges efficiently. Ultimately, this system will empower students to explore static analysis CTFs more effectively, providing them with guidance through class materials. The AI tutoring agent will help learners deepen their conceptual understanding of program analysis, vulnerability detection, and formal reasoning, without requiring constant human supervision. Instructors will benefit from reduced teaching overhead, while students will gain learning support. The long-term goal is to create confident students who can approach complex software security problems with independence and insight.

Our current proposed system includes a knowledge graph with retrieval augmented generation (RAG), which depends on course materials and relationships between static analysis concepts and CTF challenges. Students will interact with the system through a web-based chatbot, while the backend retrieves relevant concepts and enables explanations that are aligned with course objectives. Instructors have the ability to edit and expand the knowledge graph. The core system components are adapted from the technical feasibility study, and are examined in greater detail throughout this document.

Implementation Overview

The solution vision of our product, as pictured in figure 1, is an accessible chatbot tutor that uses a RAG-based pipeline to retrieve relevant information from a teacher-curated knowledge graph, which contains the documents and information needed to help students solve domain-specific problems. To achieve this, the application will be implemented using a Next.js frontend and FastAPI backend, and made accessible through any standard web browser via hosted deployment of a Docker container. Additionally, we will utilize Neo4j and PostgreSQL for knowledge graph and chat storage, respectively.

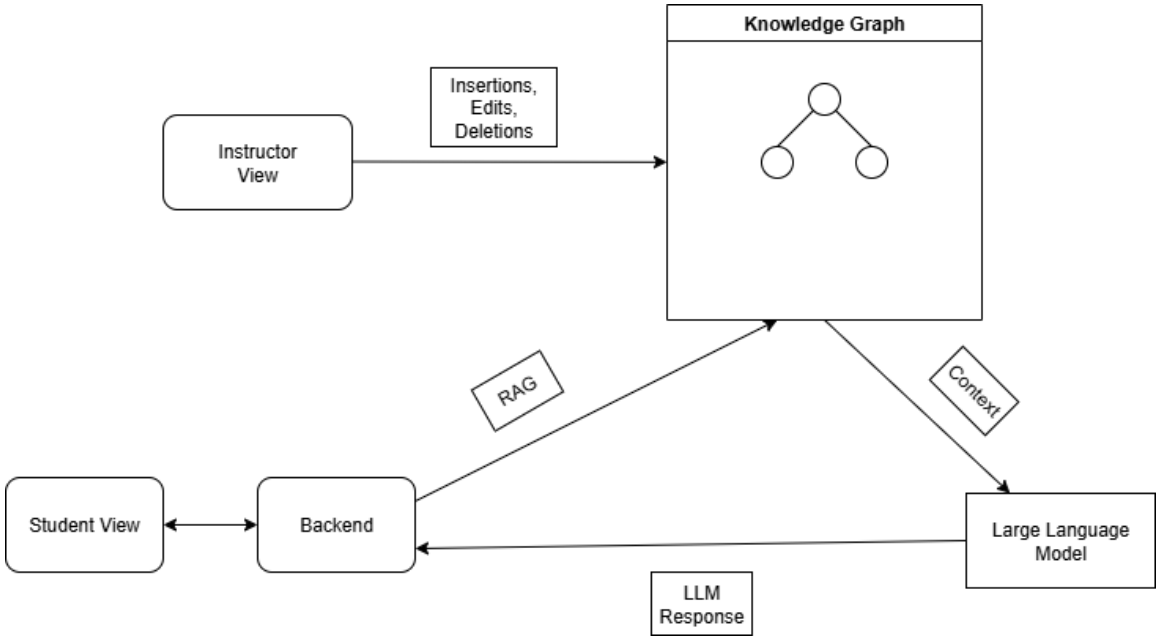

Figure 1. High-level overview of the system diagram

Architectural Overview

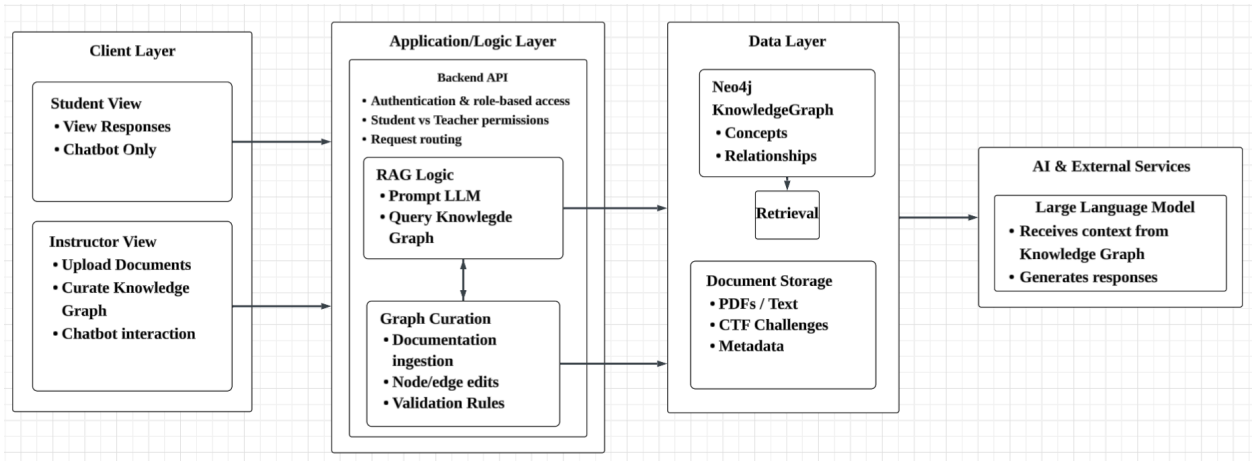

Figure 2. Architecture Diagram

From an architectural perspective, CLASS follows a multi-tier client-server architecture system composed of the following layers: client, application, data and AI and external services. This separation between layers reduces coupling between UI and backend logic, this allows scalability within each tier. The client layer consists of a browser-based frontend that provides a specific user experience depending on the user's role, either the student or instructor view. The application tier implements the system's logic, including user authentication, role-based access control, graph retrieval, and response composition. The data tier manages storage and a structured Neo4j knowledge graph, relational databases for user and session data, and document

storage. This project also implements a large language model with its application, as noted in the AI & External Services tier in figure 2. This LLM will be a part of the retrieval augmented generation pipeline. Its use within the project will act as a response generator for the explanations retrieved from the knowledge graph and the supporting documents. The LLM is invoked through our backend that queries and retrieves the nodes, assembles a prompt based on retrieved content, and generates a response. This component was chosen by our group because of the goal of having instructional alignment. We expect the responses to be based on concepts received from the class materials, and this in turn will reduce hallucinations and keep explanations consistent with course objectives. This layered design allows our project to be maintainable, scalable, and secure by isolating the layers' responsibilities.

At the user level, the system supports two primary roles, student and instructor. Students must be able to authenticate, initiate, and revisit chatbot sessions. These responses must reference static-analysis concepts and explain how techniques relate to each other. Instructors must be able to manage the knowledge base by adding, editing, and organizing concepts and relationships, ensuring that the content remains accurate throughout the course.

From these user needs, several functional and performance requirements are derived. The system must implement a graph retrieval pipeline that supplies the language model with relevant concepts and relationships for the response generation. Based on the role-based access control, instructors can modify the knowledge graph, while students maintain read-only access through the tutoring queries. To maintain usability in an instructional setting, the system must deliver responses with low latency and support concurrent users without the degradation of performance. The system is implemented as a full-stack web application, and the architecture allows us to integrate a Neo4j knowledge graph for modeling, GraphRAG for retrieval, a backend for response construction and security, and finally NextJS frontend for response and user interaction. These requirements and architectural constraints provide context for understanding the design and implementation decisions presented in the remainder of this document.

Component-Level Design

Following the architectural overview, this section discusses the internal design of each major component of the project. The previous section establishes the client, logic, data and external service layers, for component-level design, it explains the technical responsibilities implemented within those layers.

Client Layer:

The Client Layer is responsible for managing all user-facing authentication interactions and presenting role-aware UI state within the browser-based front end. This component encapsulates user sign-in, sign-up, role selection, temporary storage of authentication status and user role, and sign-out, while remaining strictly bound to the presentation logic. Its primary

modules include the UserIcon component, which acts as the entry point for authentication interactions and displays current session status, and the SignInPopup component, which manages the multi-step authentication workflow (sign-in, sign-up, and role selection). Internally, the Client Layer component is composed of these UI modules that coordinate user interface state using React hooks and save the authenticated username and user role in the browser's localStorage. For security reasons, user passwords are never stored or kept on the client. Instead, credentials are transmitted once to the backend API over HTTP, where password hashing and verification are handled server-side before the credentials are discarded by the client. The Client Layer communicates with the Backend API exclusively through JSON-based requests to the /signup and /signin endpoints and updates the interface based on the return authentication outcome and role information. This component does not perform authentication logic, password management, authorization checks, or store user credentials beyond what is needed to maintain the UI state, enforcing a clear separation of concerns and positioning the Client Layer as the boundary between end users and backend services contained in the Logic Layer.

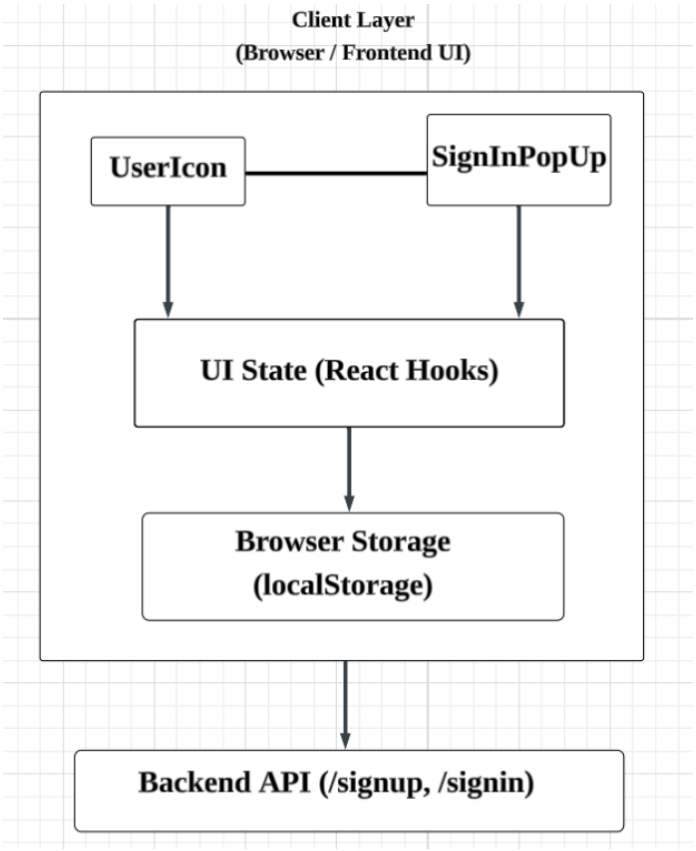

Figure 3. Client layer

Logic Layer:

The Logic Layer implements the core application functionality of the system, serving as the intermediary between the Client Layer and the Data Layer. Its primary responsibilities include authentication and role management, RAG-based response generation, and knowledge graph curation. Internally, it is organized into several key modules:

1. The authentication module manages sign-up and sign-in operations through the sign-up and sign-in endpoints, hashing and verifying passwords with the hash_password function while enforcing role-based access control.

2. The RAG module initializes the retrieval pipeline using OpenAIEmbeddings for text to vector conversion, VectorRetriever for retrieving relevant context from the documents, GraphRAG to retrieve contextually relevant knowledge from the Neo4j graph, RAGTemplate for providing the LLM with context, and OpenAI LLM for tutor response generation. Conversation history is maintained in InMemoryMessageHistory. We constrain LLM to answer as a tutor using only provided information through strategic prompt engineering.

3. The Graph Curation module is primarily implemented through kg.py, which provides functions such as get_full_graph to retrieve all node and edge information from the knowledge graph database through a Cypher query, forming the basis for instructor or system operations on the graph.

The Logic Layer exposes HTTP endpoints /signup, /signin, /generate_response, /get-graph-info that accept JSON requests and return structured responses for authentication, user roles, AI-generated messages, or graph data. It centralizes logic and role-based control, providing secure, fast responses for students while enabling instructor access to the knowledge graph, which is contained in the Data Layer.

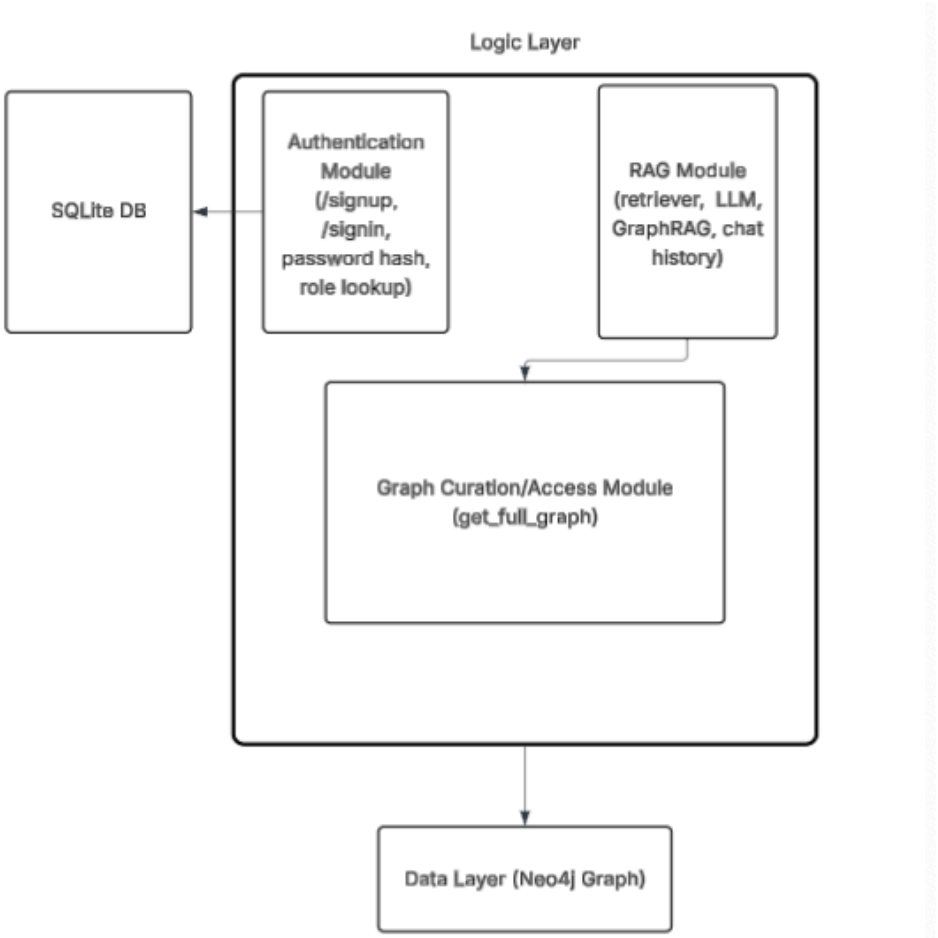

Figure 4. Logic layer

Data Layer:

Within our data layer component, the contextual archive of class materials necessary for appropriately curated model responses from the large language model (LLM) is hosted. The database contains information such as CTF challenges, solutions, PDFs, Slides, and relevant metadata. The knowledge database is hosted through Neo4J Aura, which adopts a native graph design, excelling in analyzing complex, deep, and dynamic relations in real time, which is ideal for our use case. The addition of Neo4J allows us to effectively query the graph for our complex relationships using its index-free adjacency feature. Therefore, when requests are made from our backend run through our database, the retrieved response is more efficient than standard relational-database systems. The data stored in the database is displayed and curated by teachers through the application layer, though storing and fetching all node and edge data. This layer allows for optimal usage of services that are externally hosted.

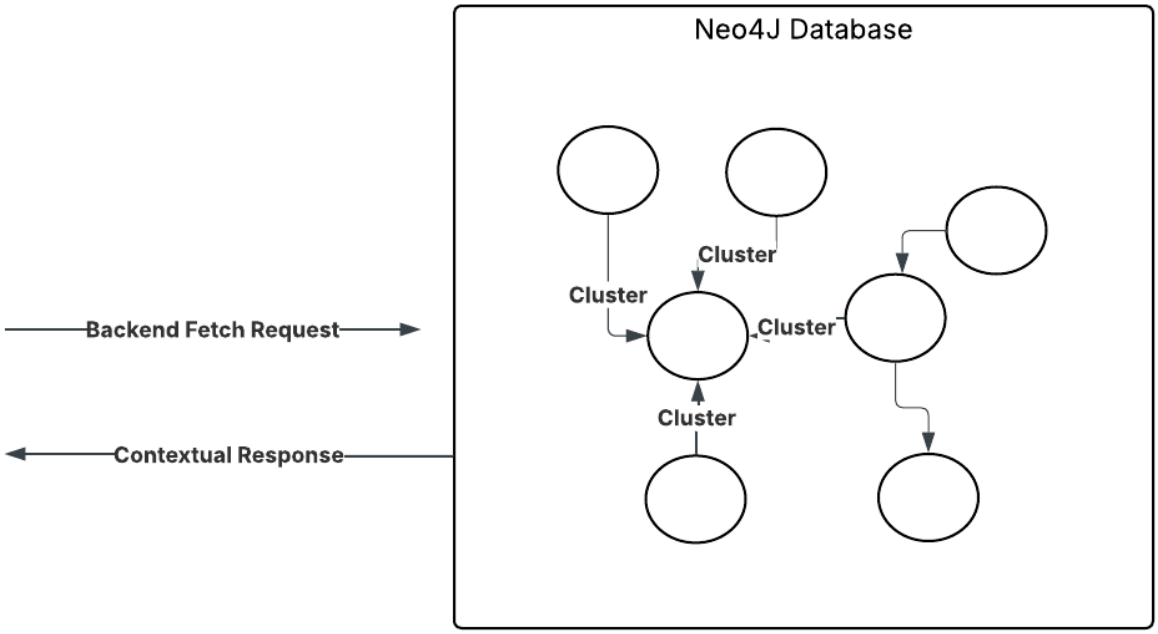

Figure 5. Data layer

External Service:

Our external services component will feature the chosen large language model and serve as the agent for which our responses will be generated. Our system will fetch a response from the model with a payload sent from our backend that includes three crucial features: User response, curated documents relevant to the request, and a prompt-engineered script to model certain behaviors within the response. The LLM will digest the payload and return an appropriate response to the user that will be displayed back on the frontend.

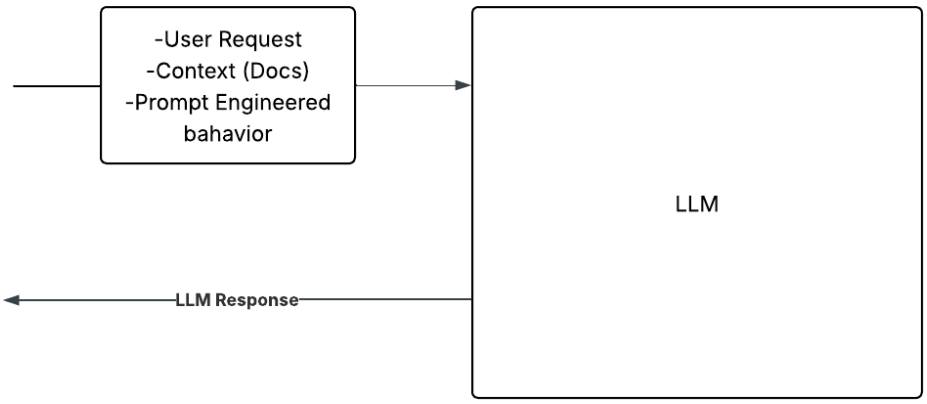

Figure 6. AI and External Services

Implementation Plan

After establishing the system architecture and component-level design, this section will outline a plan to implement each component. The following timeline outlines the phases of major components, and the development and integration milestones.

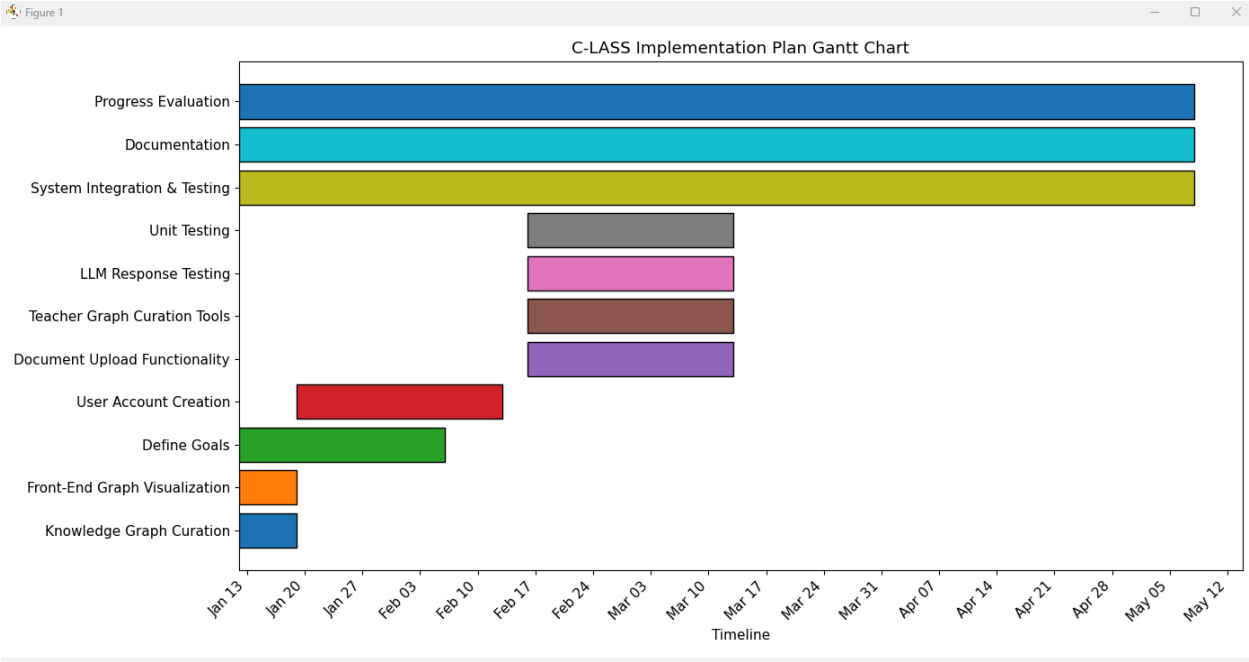

Figure 7. Implementation Plan

The implementation timeline is organized into two primary development phases followed by an evaluation period. Phase 1 focuses on establishing the project foundation, including goal definition, initial knowledge graph curation, front-end graph visualization, and user account creation. Phase 2 builds on this foundation by introducing higher-level system functionality, including document upload capabilities, teacher-facing graph curation tools, large language model (LLM) response testing, and unit testing. These tasks are scheduled concurrently to support iterative integration. The evaluation phase spans the full project timeline and includes ongoing testing, system integration, documentation, and progress evaluation. This phased approach enables parallel work across team members while managing dependencies between the core infrastructure.

The main technical risk of our system is the volatility of LLMs. Although RAG and prompt engineering is expected to mitigate hallucination, there are still prompt injection risks. We plan to combat this through sanitization and validation layers. Additionally, our reliance on Neo4j libraries and wrappers introduces a dependency risk, so our application is prone to becoming outdated if their systems are no longer supported. Finally, the linear cost scaling of hosting LLM and database reliant projects, such as this one, poses a limitation to this project. We

do not have the funds to host this application in a production setting, so it will be designed as a proof of concept.

Conclusion

As we iterated throughout the document, CLASS is a tutoring platform that aims to support students in need of supplemental guidance for demanding course loads. CLASS offers an adaptable platform that enables students to digest complex concepts, such as CTF challenges, enforcing core concepts within students before code is executed. Our features will enable the model to behave in a manner that is accustomed to a teaching assistant, allowing students to receive guidance and inquisitive feedback beyond standard school hours. With our sponsor, our approach is tailored to Cybersecurity with a focus on delivering instructional guidance to students within difficult CTF challenges that enable students to learn. The project delivers proper instruction through instructor-curated documentation and relations that are used contextually within our model. The RAG system will allow model responses to be more precise on the complexities found within the SE450 coursework, with instructor context supervision. Current interactions of our proposed solution include the chat interface, featured along with an integrated database context retrieval. Additionally, we have implemented a feature to visualize our graph for instructor viewing, with features to edit the graph to come. Overall, CLASS aims to deliver a product that benefits students with several applicable materials.