TED UNIVERSITY

CMPE491/SENG491 Senior Project

SyntaxSavior Project High-Level Design Report

27.12.2024

Team Members:

İrem BEŞİROĞLU Software Engineering

Yiğit Oğuzhan KÖKTEN Computer Engineering

Yüksel Çağlar BAYPINAR Computer Engineering

Contents

1. Introduction

The High-Level Design presents the architectural approach and system components that shall be necessary to realize a robust, user-centric platform integrated with advanced technologies like an informational panel plug-in, safe-exam-browser, and vector database. These will enhance user experience, ensuring that the interactions within the system are secure and efficient.

1.1. Purpose of the System

The system shall henceforth provide an integrated one-stop platform to support students, staff, and faculty with intelligent and secure tools. An informational chatbot will be used to bring relevant information to them as fast as possible. The safe-exam-browser ensures a secure environment for conducting online assessments, while the vector database facilitates efficient retrieval and management of large datasets. Two multi-agent models for further advancement in functionality shall also be integrated into the system:

- **Code Analysis Agent**: This model does code analysis and sends the findings to the server.
- **Topic Retrieval Agent**: The server evaluates the code analysis findings, pulls the appropriate course topics from the dataset, and smoothly integrates them.

1.2. Design Goals

- **Scalability:** The system should be capable of handling increased loads, such as user inquiries, data retrieval, and examination traffic.
- **Security:** By using a safe-exam-browser, the system assures that assessments are secure, preventing unwanted access and cheating.
- **Efficiency:** The informational panel plug-in and vector database must respond quickly and accurately, enhancing the entire user experience.
- **Intelligence:** Multi-agent models should provide smart insights, such as code analysis and contextual subject recommendations, to successfully satisfy user demands.
- **User-Centric Design:** The interface and operations should prioritize usability for students, staff, and instructors.

1.3. Definitions, Acronyms, and Abbreviations

- **Safe-Exam-Browser (SEB):** A customized web browser built for safe online exams.
- **Vector Database:** A database designed to handle vector-based data, including similarity searches and quick retrieval.

- **Multi-Agent Models:** Independent AI components that collaborate with one another and the server to complete specific tasks.
- **Code Analysis:** The process of reviewing and evaluating source code for functionality and errors.
- **Course Topics Dataset:** A systematic collection of course-related data for recommendation and retrieval purposes. This collection is collected from the CMPE113 course syllabus and coding-related pages on the Internet.

1.4. Overview

The system integrates three primary components:

1. **Informational Panel Plug-in:** A non-communicative chatbot that provides quick access to commonly sought information.
2. **Safe-Exam-Browser:** A safe platform for conducting fair and regulated online examinations.
3. **Vector Database:** A high-performance storage system designed for efficient data management and retrieval.

Furthermore, the two multi-agent models work together to improve functionality:

- **The Code Analysis Agent:** Examines user-supplied code and reports the findings to the server.
- **The Topic Retrieval Agent:** Evaluates these findings, searches the course topics dataset, and makes appropriate suggestions to users.

This High-Level Design Report lays out a thorough plan for attaining these objectives while ensuring the system fulfills performance, security, and usability standards.

2. Current Software Architecture

The existing system utilizes Virtual Programming Lab (VPL), integrated within the university's Learning Management System (LMS). Students develop their code using Eclipse IDE and submit completed assignments through VPL's web interface, accessed via their university credentials.

VPL operates as an automated grading system, executing submitted code against predefined test cases with specified inputs and expected outputs. This architecture creates a linear workflow:

1. Development in Eclipse IDE
2. Manual code file upload to VPL through LMS
3. Automated execution and grading based on test cases
4. Results displayed to both students and instructors

This system, while functional for basic grading purposes, lacks real-time feedback capabilities and integrated learning support during the development process. The disconnected nature of the development environment (Eclipse) from the grading system (VPL) creates a gap in the learning workflow where students cannot receive immediate guidance on their code.

3. Proposed Software Architecture

3.1. Overview

SyntaxSavior employs a modular, distributed architecture consisting of three primary subsystems: an IDE plugin, a central backend processing system, and a frontend user interface. The architecture follows a client-server model with event-driven communication, where the IDE plugin acts as the primary data source, the backend handles complex processing and analysis, and the frontend provides visualization and user interaction capabilities.

3.1.1. IDE Plugin Subsystem

The Eclipse-integrated plugin serves as the primary interface for code monitoring and initial analysis. It implements real-time syntax checking and basic error detection through integration with Eclipse's native diagnostic tools. The plugin maintains bidirectional communication with the frontend interface, sending code updates and receiving formatted feedback for display within the IDE environment. This subsystem handles all direct code access and preliminary analysis, ensuring minimal performance impact on the IDE while providing essential data to the larger system.

3.1.2. Central Backend Processing Subsystem

The backend operates as a locally hosted server implementing the core analysis logic and AI integration. It receives code submissions from the frontend, processes them through both traditional analysis tools and a custom-trained language model, and generates comprehensive feedback based on curriculum-aligned criteria. This subsystem maintains a structured database of course materials and common programming concepts, which it uses to contextualize and enhance its feedback generation. The language model integration is specifically designed to provide educational guidance rather than direct solutions.

3.1.3. Frontend Interface Subsystem

The frontend provides a lightweight, dynamic interface for displaying processed information and managing user interactions. It serves dual purposes: presenting analyzed code feedback and providing access to structured course materials. The interface communicates with both the IDE plugin for code submission and the backend for receiving processed analysis results. This subsystem emphasizes minimal resource usage while maintaining responsive performance and clear information presentation.

3.2. Subsystem Decomposition

Each major subsystem can be further decomposed into specialized components:

3.2.1. IDE Plugin Components

- **Code Monitor:** Interfaces with Eclipse's editing environment to track code changes
- **Syntax Analyzer:** Integrates with existing linting processes for basic error detection
- **Communication Handler:** Manages bidirectional data flow with the frontend
- **UI Integration Manager:** Handles plugin-specific interface elements and popups
- **Event Dispatcher:** Coordinates plugin events and user interactions

3.2.2. Backend Processing Components

- **Request Handler:** Manages incoming analysis requests from the frontend
- **Code Analysis Engine:** Processes code for logical and structural issues
- **Language Model Interface:** Manages communication with the custom AI model
- **Curriculum Database:** Stores and manages course materials and concept explanations
- **Response Generator:** Formats analysis results for frontend consumption
- **Session Manager:** Handles user session tracking and state management

3.2.3. Frontend Interface Components

- **Data Display Manager:** Handles visualization of analysis results
- **Course Material Navigator:** Provides interface for accessing lecture materials
- **Communication Controller:** Manages data flow with backend and plugin
- **State Manager:** Maintains frontend application state
- **UI Component Library:** Collection of reusable interface elements

This decomposition enables:

- Clear separation of concerns between system components
- Scalable integration of future features like Safe Exam Browser
- Maintainable codebase with well-defined component boundaries
- Efficient resource utilization through component specialization

- Flexible deployment options for varying devices of both students and the university

3.3. Hardware/Software Mapping

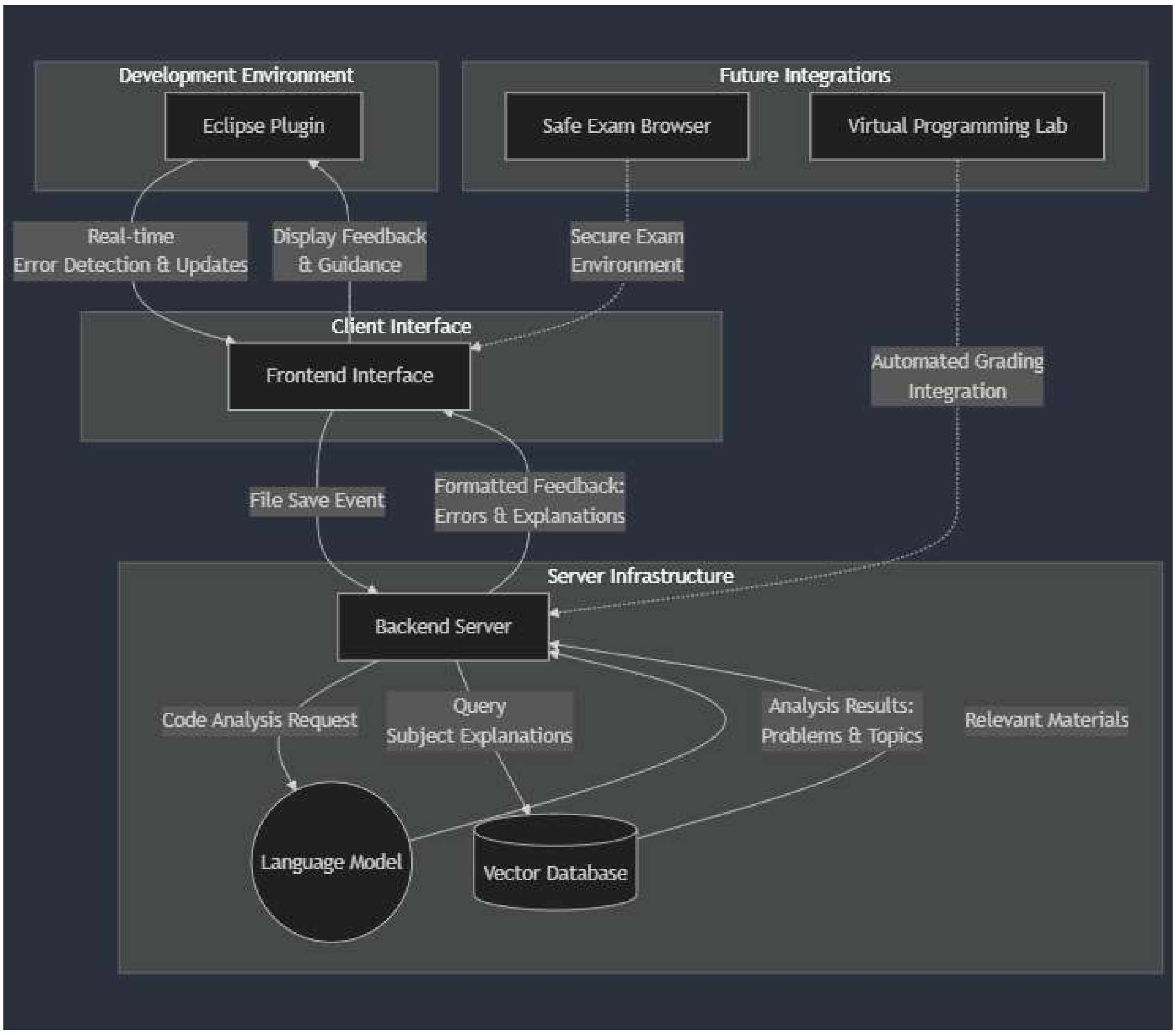

Figure 1: Drafted Hardware/Software Mapping for SyntaxSavior

3.4. Persistent Data Management

The whole strategy of the persistent data management is based on how to utilize the vector database for efficient and scalable data retrieval with awareness of the context. Based on this strategy, main functionalities can be supported, like chatbot interactions, recommendations of course topics, and integration of code analysis.

3.4.1. Purpose of Persistent Data Management

The vector database is designed to:

- Store semantic data representations (e.g., course materials, code analysis results, and FAQs) as high-dimensional vectors for similarity-based searching.
- Allow for speedy and accurate retrieval of relevant information depending on user queries or system inputs.
- By dynamically obtaining appropriate course subjects or instructional material, you may help multi-agent interactions get contextually grounded.

3.4.2. Key Components

3.4.2.1. Vector Database:

- The system will utilize a vector database such as Milvus, ChromaDB or MongoDB, which is optimized for handling vector embeddings generated by machine learning models.
- The language model (API that will be created later) and other agents generate embeddings, which represent both textual and structured data.

3.4.2.2. Data Sources:

- **Course Dataset:** The dataset includes course materials, learning objectives, and syllabus data that may be matched to student needs.
- **Code Analysis Results:** The code analysis agent's outputs are saved as indexed vectors for future reference.
- **Frequently Asked Questions:** Captures frequently requested questions and their embeddings to help the chatbot provide timely and correct replies.

3.4.2.3. Storage and Indexing:

- All data stored in the vector database will be indexed using similarity metrics (e.g., cosine similarity) to ensure rapid retrieval of the most relevant content.
- Periodic indexing and retraining methods will be used to accommodate fresh data entries and enhance system correctness.

3.4.2.4. Integration with Multi-Agent Models:

- The code analysis agent creates embeddings of code outputs and sends them to the server, where they are stored in the vector database.
- When the informational panel plug-in or server needs information, such as course subjects or related content, it sends a query to the vector database, where the most relevant results are returned.

3.4.3. Design Considerations

3.4.3.1. Scalability:

- The vector database is designed to manage enormous amounts of data while maintaining low-latency queries as the system grows with new datasets.

3.4.3.2. Security:

- The embedding storages will be encrypted when including sensitive data, such as student queries or personal information, to the database.
- Only valid agents or users will have permission via some control of role-based access interaction over the database.

3.4.3.3. Performance Optimization:

- Efficient indexing methods reduce query latency.
- To keep the database healthy, garbage collection should be performed on a regular basis.

3.4.3.4. Data Integrity:

- Backup and recovery techniques protect data in the case of a system breakdown.

3.4.3.5. Flexibility:

- The vector database structure facilitates the incorporation of new datasets or changes to old ones, enabling long-term flexibility.

3.4.4. Workflow

3.4.4.1. Data Ingestion:

- The language model or code analysis agent transforms raw data, such as course materials or code analysis findings, into embeddings.
- The vector database stores embeddings together with metadata tags for categorization and retrieval.

3.4.4.2. Query Processing:

- When a query is launched (for example, by a chatbot user or the multi-agent system), the vector database does a similarity search to find the most closely related embeddings.
- The results are graded and returned to the requesting module or agent for further processing.

3.4.4.3. Updates:

- To keep the system up to date, the vector database is updated on a regular basis with new course subjects, FAQ updates, and better code analysis outputs.

3.4.5. Benefits of Using a Vector Database

- **Contextual Understanding:** The system may deliver accurate, context-driven replies by accessing semantically related material.
- **Efficiency:** Handles high-dimensional embeddings and provides fast similarity searches.
- **Adaptability:** Enables dynamic changes and scaling in response to the system's increasing requirements.

By incorporating a vector database, the system ensures efficient, accurate, and scalable data management that aligns with the high-level design goals.

3.5. Access Control and Security

3.5.1. Authentication

The system will implement secure user authentication protocols to restrict access and prevent unauthorized use. Access will be granted exclusively to users with verified email addresses in the ‘tedu.edu.tr’ domain, ensuring that only legitimate members of the institution can use the platform. Additionally, integration with the existing TeduPass authentication system will be explored to streamline the user onboarding process, enabling users to authenticate using their existing LMS accounts with no additional steps.

3.5.2. Authorization

The system will define four distinct user roles: student, instructor, assistant, and administrator, each with specific privileges and access levels.

- **Students** will have access solely to their own projects and personalized feedback, ensuring privacy and data security.
- **Instructors** will have the ability to monitor student progress, view system usage statistics, and configure specific settings within the system.
- **Assistants** will share the ability to monitor student progress and view system usage statistics, in addition to the ability to upload lab documents and solution guides to help pre-organize system use.
- **Administrators** will possess full access to system configuration, user management, and data management functions, ensuring comprehensive oversight and control over the platform.

3.5.3. Data Privacy and Security

The system will implement stringent measures to ensure the confidentiality and security of all data, including student code, personal information, and interaction records. All data transfers within the system will be encrypted to protect against unauthorized access. Additionally, submissions will be hashed to verify their validity and authenticity, ensuring that code integrity is maintained, safeguarding both from transmission artifacts, and malicious tampering. Appropriate safety measures will be put in place to prevent data breaches and unauthorized access.

3.5.4. Anti Cheating Measures

To maintain academic integrity and prevent misuse of the system, several measures will be put in place:

- The frequency of hint requests will be restricted to prevent over-reliance on the system for answers.
- The platform will be designed to avoid providing any direct code solutions to students.
- Code similarity detection will be implemented to identify potential instances of academic dishonesty, such as plagiarism.

- User interactions with the system will be monitored for suspicious patterns of behavior that could indicate attempts at abuse.

These measures will be carefully calibrated to avoid penalizing students unjustly while ensuring that the system remains fair and secure.

3.6. Global Software Control

3.6.1. System Configuration

A central management interface will be provided for system administrators to configure and manage key system parameters. These include:

- Defining error severity levels and the corresponding feedback mechanisms.
- Setting the frequency and content of hints provided to users.
- Managing the available learning resources and determining how they will be presented to users.
- Configuring access control and security settings for the system.
- Administering system-wide notifications and announcements to keep users informed.

3.6.2. Version Control and Updates

A version control system will be implemented to manage and deploy updates to the SyntaxSavior platform.

- Updates will be scheduled to avoid conflicts with ongoing lab sessions, ensuring that user activities are not disrupted.
- Procedures will be established for rolling back updates if unforeseen issues arise during or after deployment.

3.6.3. Monitoring and Logging

Comprehensive monitoring and logging mechanisms will be implemented to ensure the system operates smoothly:

- The system will be continuously monitored for performance, with logs and live resource tracking used to identify potential issues.

- User behavior and system usage patterns will be analyzed to gain insight into system interactions.
- Diagnostic tools will be used to efficiently detect and resolve any problems that arise.
- Log data will be securely stored and used for ongoing system improvement and maintenance.

3.7. Boundary Conditions

3.7.1. Error Handling

Robust error handling mechanisms will be employed to ensure the system responds gracefully to unexpected situations, including:

- Network connectivity disruptions.
- Resource limitations within the system.
- Invalid user input.
- Unanticipated exceptions occurring within the system.

Informative error messages will be provided to both users and administrators to guide them in resolving issues promptly.

3.7.2. Edge Cases

The system will be designed to address the following edge cases:

- **Special Holidays or Events:** Unscheduled breaks, such as national holidays or university-specific events, may interfere with the predefined curriculum schedule, potentially delaying or disrupting planned assignments and exams. The system should adapt by automatically adjusting deadlines or rescheduling sessions in such cases.
- **Curriculum Adjustments Mid-Term:** Modifications to the curriculum, such as changes in course content or deadlines during the semester (e.g., due to unforeseen circumstances or academic calendar adjustments), could require updates to the system's schedule, learning materials, or assessments.
- **Varying Attendance Levels in Lab Sessions:** Fluctuating attendance in lab sessions, where some students may be absent or participate asynchronously, can result in a disparity in real-time feedback and progress tracking. The system will need to accommodate students with varying levels of participation and ensure that feedback remains consistent and timely for all users.

- **Unexpected Instructor Absence or Change:** If an instructor is unexpectedly absent or replaced mid-semester, students may require additional support or clarification regarding ongoing assignments, which may disrupt the typical feedback cycle. The system will include features to notify users of such changes and offer alternative support resources.
- **Server Overload During Peak Periods:** During peak periods, such as right before assignment submissions or exams, the system may experience high traffic, potentially affecting performance. The system will need mechanisms to handle increased loads without impacting user experience, such as load balancing or temporary reduction in non-essential services.
- **Students' Personal Circumstances:** Students facing personal issues, such as illness or family emergencies, may require extensions or adjustments to assignment deadlines. The system should allow for personalized exceptions to be made, either through instructor intervention or automated processes that provide flexibility.

3.7.3. Scalability and Performance

The system will be engineered to accommodate a high number of concurrent users and an increasing volume of data.

- Performance optimization will be a priority to ensure system responsiveness and minimize latency.
- Regular performance monitoring will be conducted to assess system load, and resources will be adjusted as necessary to maintain optimal performance.

4. Subsystem Services

4.1. IDE Plugin Subsystem Services

The **IDE Plugin** subsystem serves as the primary interface for code development and analysis within Eclipse. Its key services include:

- **Real-Time Code Monitoring and Syntax Checking**: The plugin will continuously monitor code changes in the Eclipse environment, providing immediate syntax analysis and basic error detection. This service enhances the development experience by offering instant feedback to students, allowing them to correct errors as they write code.

- **Code Submission and Feedback**: Once a student submits their code through the plugin, the system will transmit the submission to the backend for further analysis. The plugin will also receive feedback from the backend, presenting it within the IDE to the user in an easy-to-understand format. This feedback will include identified errors, suggestions for improvements, and any relevant course material for context.

- **Bidirectional Data Communication**: The plugin will facilitate communication between the frontend and the backend, ensuring seamless data exchange for code submissions, feedback, and system alerts. The plugin ensures that the data flow between the IDE and the other subsystems remains secure and efficient.
- **Real-Time Error Detection and Suggestions**: The plugin will not only identify syntactical errors but also provide basic suggestions and hints based on common coding issues, which students can review for immediate guidance.

4.2. Central Backend Processing Subsystem Services

The **Backend** subsystem acts as the core processing unit for analyzing student submissions and generating feedback. It provides several critical services:

- **Code Analysis and Evaluation**: After receiving code submissions from the IDE plugin, the backend will analyze the code for logical, structural, and syntactical issues. This service uses both traditional analysis tools and a custom-trained AI model to ensure comprehensive evaluation based on curriculum-aligned criteria. The analysis includes identifying errors, offering insights on best practices, and recommending improvements.
- **Curriculum-Aligned Feedback Generation**: The backend will generate feedback contextualized to course materials and topics. This feedback will be directly tied to the CMPE113 course syllabus, offering students advice that is relevant to their current level of understanding and the course's objectives.
- **Language Model Interaction**: The backend will integrate with a custom AI language model to provide educational guidance rather than direct solutions. This model will offer insights based on the analysis of the code and the curriculum, enhancing students' understanding of programming concepts.
- **Data Management and Storage**: The backend will handle the storage and retrieval of course materials, including programming resources, course topics, and student progress records. It will also manage user authentication, access control, and session tracking.
- **Error Handling and Reporting**: The backend will process errors and handle situations where code analysis cannot be completed. It will generate appropriate error messages, log system anomalies, and notify administrators if necessary.

4.3. Frontend Interface Subsystem Services

The **Frontend Interface** subsystem provides the user-facing interface for interacting with the system. The services it offers include:

- **Visualization of Code Analysis Results**: After the backend processes the student's code, the frontend will display the analysis results in an easy-to-read format. This will include details on errors, warnings, and suggestions for improvement, as well as links to relevant course materials for further learning.
- **Access to Course Materials**: The frontend will provide an intuitive interface for accessing lecture notes, tutorials, and other course-related materials. Students can navigate these materials alongside their code submissions, promoting a more integrated learning experience.
- **User Interaction Management**: The frontend subsystem will be responsible for managing user interactions within the system. This includes presenting alerts, notifications, feedback, and ensuring that the system remains responsive to user inputs.
- **Seamless Data Communication**: The frontend will manage data flow between the IDE plugin and the backend, ensuring that code is correctly submitted, feedback is retrieved, and users receive necessary alerts or updates. It will also provide users with real-time updates on the status of their code analysis.
- **User Session Management**: The frontend will keep track of user sessions, ensuring that users remain logged in and can access their previous interactions and submissions. This service will also help manage timeouts and session expiration for security purposes.

4.4. Additional Services

Beyond the core services of the subsystems, additional functionality will be integrated into the system to enhance its performance and user experience:

- **Security Services**: As discussed in the access control and security section, the system will implement security services such as encrypted data transmission, multi-factor authentication (MFA), and audit logging to monitor system interactions and safeguard user data.
- **Anti-Cheating and Abuse Prevention Services**: In line with the anti-cheating and abuse prevention measures, the system will offer services such as:
  - **Code Similarity Detection**: The backend will use algorithms to compare code submissions and identify potential instances of plagiarism.

- **Monitoring and Pattern Recognition**: The system will analyze user behavior for suspicious patterns, flagging potential abuse such as excessive hint requests or irregular code submission patterns.
- **Usage Monitoring**: The system will track the frequency and nature of hint requests, ensuring that students use hints responsibly without relying on them excessively.

5. Glossary

5.1. Core Components

- **SyntaxSavior**: The system designed to assist students in learning programming through real-time feedback, code analysis, and course resources.
- **IDE Plugin**: A software extension integrated into Eclipse to provide real-time syntax checking, code analysis, and feedback.
- **Frontend Interface**: The user-facing part of the system that displays code analysis results, course materials, and facilitates communication between the plugin and backend.
- **Backend**: The central server that processes user requests, handles code analysis, and manages data storage.
- **Language Model**: A machine learning model integrated into the backend to analyze student code and provide educational feedback.
- **Curriculum Database**: A repository of course materials, syllabi, and explanations of programming concepts.
- **Code Analysis Agent**: An AI-powered agent that analyzes user-submitted code and sends findings to the server for processing.
- **Topic Retrieval Agent**: An AI agent that retrieves relevant course topics based on code analysis findings.

5.2. Data and Communication

- **Bidirectional Data Communication**: The two-way flow of data between frontend and backend systems.
- **Embedding**: A high-dimensional vector representation of data (e.g., text, code) used for similarity-based searching.

- **Vector Database**: A database storing data in vector form, enabling fast similarity searches based on embeddings.
- **Similarity Search**: A query process for finding semantically similar data in the vector database.
- **Data Ingestion**: The process of converting data into embeddings for storage in the vector database.
- **Plagiarism Detection**: Mechanism for identifying code similarity to detect potential academic dishonesty.
- **Version Control**: System for managing changes to code or documentation over time.
- **Feedback and Learning:**
- **Curriculum-Aligned Feedback**: Feedback tailored to the course content and objectives.
- **Data Display Manager**: A frontend component responsible for visualizing analysis results and feedback.
- **Learning Management System (LMS)**: A platform for hosting course content, assignments, and managing student-instructor communication.
- **Virtual Programming Lab (VPL)**: An automated grading system integrated with the LMS for code submission evaluation.

5.3. Security and Access

- **Role-Based Access Control (RBAC)**: A method to restrict access based on user roles within the system.
- **User Authentication**: The process of verifying user identity before granting access.
- **Access Control**: Processes that manage who can access specific data and system resources.
- **Security Services**: Measures like encryption and multi-factor authentication to protect system data.

5.4. System Management

- **Server Overload Management**: Strategies like load balancing and scaling to handle high traffic.
- **Session Manager**: Tracks and manages user sessions across the platform.
- **State Manager**: Maintains the current state of the frontend application.

- **Process and Workflow:**
- **Error Handling**: Processes that ensure issues are identified and resolved to minimize system disruption.
- **Logging and Monitoring**: Continuous tracking of system activities for troubleshooting and maintenance.
- **Workflow**: The sequence of processes for data ingestion, analysis, and result delivery.
- **Event Dispatcher**: Coordinates user interactions and plugin-specific events.

5.5. Additional Components

- **Safe-Exam-Browser (SEB)**: A secure browser for online assessments.
- **Informational Panel Plugin**: A non-communicative chatbot providing quick access to frequently sought information.
- **Communication Controller**: Manages data flow between frontend and other subsystems.
- **Communication Handler**: Facilitates data exchange between the plugin and frontend interface.
- **Session Manager**: Tracks and manages user sessions to maintain interaction continuity.

6. References

1. CMPE 113: Introduction to Programming Syllabus. TED University, [2024].
   Course syllabus for the CMPE 113 Introduction to Programming class, providing foundational information on course structure, objectives, and expectations.

2. CMPE 491: Senior Project 1 Syllabus. TED University, [2024].
   Course syllabus for the CMPE 491 Senior Project 1 class, outlining guidelines and expectations for senior-level project work, relevant to the development of the SyntaxSavior tool.

3. Mermaid Live Editor, Accessed at: https://mermaid.live/