TED UNIVERSITY

Department of Computer Engineering

CMPE 491 High Level Design Report

Team Members:

- Arda Taha Sökelen (10342323378)
- Mehmet Alp Demiral (34937067942)
- Miray Aday (13849221628)
- Oğuzhan Altın (15052066386)

Supervisor: Hakkı Gökhan İlk

Jury Members:

- Gökçe Nur Yılmaz

TABLE OF CONTENTS:

1. Introduction

1.1 Purpose of the system

The principal aim of **Project SAGE** (Smart AI Guide for Education) is to assist TED University students with their academic processes and administrative processes in a more productive manner using artificial intelligence technologies. This system was designed to support real-time, tailored and contextualized assistance to students as they make their way through the academic process with an emphasis on improving course selection, academic scheduling and student engagement.

One of the most significant challenges students face during course registration is easy and reliable access to accurate course information. There is a significant lack of centralized access to detailed course information. Students utilize informal and fragmented sources of information, such as social media groups, and ask friends and peers for guidance. These sources of information often lead to inaccurate information and confusion. Project SAGE will serve as a solution to this problem by creating an intelligent chatbot that can access syllabus data, read it, and help students make informed academic decisions based on reliable information sourced from the institution.

Moreover, the system strives to assist students in productivity and accountability by automating calendar integration, facilitating calendar submission. SAGE analyzes students' university email accounts (with student consent only) to determine significant academic events such as exams, quizzes, project due dates, and meetings. These events are linked to students' own Google calendars. With this feature, less mental capacity is needed from students and they are reminded of important dates without requiring their input.

In addition to tasks or activities that involve student learning, SAGE has a broad function that is to assist students, especially newcomers and international students, with social and administrative aspects of university life. Helping students find important places on campus, and providing information about clubs, events, and administrative processes is all part of the chatbot functionality which allows for a more inclusive and open-campus experience.

Ultimately, the system is envisioned to be a modular, scalable, and updatable AI-powered assistant that can change alongside students and institutions. By leveraging **state of the art** (SOTA) AI models, secure data management and user-centered designs, Project SAGE is expected to be a holistic educational companion that increases student satisfaction, academic performance and utilization of university services.

1.2 Design goals

The main design objective for Project SAGE was to build a strategic, usable chatbot specifically designed to assist TED University students and aid their academic life. The system is designed to be usable by all students at TED University, regardless of personal or technical background, with a straightforward UI/UX approach, bilingual English and Turkish, and a response experience that is valuable in a web and mobile experience. To tie into inclusivity for all students, accessibility interventions for students with disabilities will be considered during development.

Another significant objective is to provide reliable, current, and contextualized responses, which will be enabled by the use of a Retrieval-augmented Generation (RAG)**[1]** model to pull relevant information from course syllabus, announcements, students emails (with consent, of course). In this way, the chatbot will be able to produce answers to course questions, timelines, and academic processes effectively and accurately.

Scalability and modularity are further design objectives. Project SAGE is intended to provide a solution for thousands of users concurrently without degrading performance. A modular design will facilitate the addition of new functionalities, including support for new languages or existing learning management systems for the institution. This safeguards that the system can step into the future and be adjusted due to new institutional needs.

Security, privacy, and ethical compliance is an important component of the project. The system will put in place robust data protection standards, including encryption of information, role based access control, consent from the user to scan their email for information. The system will comply with the regulatory frameworks in place, such as the **Turkish Personal Data Protection Law**, also known as **KVKK**, through responsible data handling and for the long term trust in the integrity of this system.

1.3 Definitions, acronyms, and abbreviations

AI (Artificial Intelligence): Refers to the simulation of human intelligence processes by computer systems. In Project SAGE, AI is utilized to generate intelligent responses to student queries based on academic data.

Chatbot: A software application designed to simulate conversation with human users. The chatbot in this project interacts with students through natural language to support academic tasks.

RAG (Retrieval-Augmented Generation): A model that combines traditional information retrieval with generative language models to produce contextually accurate and fact-based responses.**[1]**

TEDU: TED University, the institution for which Project SAGE is developed.

KVKK (Kişisel Verilerin Korunması Kanunu): The Turkish Personal Data Protection Law, which governs how personal data must be handled and protected.

Google Calendar API: An interface provided by Google that allows developers to integrate scheduling and calendar functionalities into their applications. Used in this project to automatically populate student calendars.

Email Scanning: The process of analyzing a user's inbox (with explicit permission) to identify and extract academic events such as exams, deadlines, and meetings.

User Profile: A structured collection of user-specific data, including interests, academic history, and preferences, used to personalize chatbot interactions.

Web Scraping: A method used to extract data from websites. In this project, it is applied to gather academic information from TEDU's digital platforms.

LMS (Learning Management System): A software system used by educational institutions to manage course materials, assignments, and grades. Potential integration with TEDU's LMS may be considered in future iterations of the project.

1.4 Overview

This document provides the high-level design of Project SAGE (Smart AI Guide for Education), an AI-powered chatbot system designed to help TED University students manage their academic and campus obligations. The overall goal is to improve student experience through intelligent, timely, and personalized responses to requests for support with course selection, academic events, and institutional information.

The report first provides an overview of the goals and objectives of the system and defines critical terms and concepts. Then system architectures are provided for both the current and proposed software design, functional and non-functional requirements, discussion on components of the system, storage of and access controls to data, and security. Each of the system components or subsystems and their delineated responsibilities have been provided with sufficient detail to convey how the system should operate as a holistic system.

The report, in later sections, examines how Project SAGE can provide its services through a system of modular units, and highlights the need for making scalability, privacy compliance, and sustainability a priority. Use case models, class diagrams modeling entities, interaction flows, and user interface considerations have been provided to inform design and usability considerations.

The report has been developed with the aim to provide a baseline for the development, implementation, and evaluation phases of the project. The development phase would be informed by formal system architecture and design decisions in a way that will help ensure that TEDU students' needs are addressed in an academically sound, socially responsible, and technologically feasible manner.

2. Proposed software architecture

2.1 Overview

The proposed architecture for Project SAGE adopts a modular, layered design centered around a fine-tuned local large language model (LLM) to deliver intelligent, context-aware academic support to students. Each subsystem is responsible for a specific operational domain, contributing to the overall functionality, maintainability, and scalability of the platform. The system avoids reliance on external AI services and instead emphasizes local processing, secure data handling, and tight integration with institutional resources.

At the core of the architecture is the LLM Inference Engine, which is responsible for generating coherent and accurate responses based on context-rich prompts. These prompts are prepared by the Retriever and RAG Orchestrator, which orchestrates the interaction between user queries, the Document Store (for structured academic data), and the Vector Store (for semantic search across unstructured content). The Embedding Service plays a foundational role in this pipeline by converting all relevant textual inputs into vector representations, facilitating semantic retrieval and ranking.

Surrounding these AI-focused components are operational and integration subsystems that ensure data flow and usability. The Data Synchronization module continuously ingests and updates information from diverse sources, including the university's website, syllabus documents, and student emails (with consent). Structured data is stored in a relational Document Store enhanced with pgvector, while high-dimensional embeddings are managed in a dedicated Vector Store (e.g., ChromaDB). This layered data management supports both keyword and semantic-based information access.**[2]**

User interaction is managed through a multilingual Chat & Organizer subsystem, which ensures smooth communication with the user, manages conversation history, and interfaces with the Collector Agent. The Collector Agent integrates with student emails and Google Calendar, allowing automatic detection and scheduling of academic events. These user-facing components are accessed through a secure API Gateway and intuitive Web UI that supports authentication, input validation, and seamless feedback loops.**[3]**

Finally, a dedicated Monitoring, Logging, and Alerting subsystem ensures system robustness and maintainability by tracking the performance of all components, recording system events, and notifying administrators of failures or anomalies. This comprehensive and locally managed architecture enables Project SAGE to function as a privacy-conscious, extensible, and intelligent assistant tailored specifically to TED University students' academic needs.

2.2 Subsystem decomposition

What to do in Subsystem Decomposition

Synchronization

The data synchronization module of the Project SAGE system is one of the key components that keeps the system up-to-date. This component collects data from university systems, student emails, bulletin boards and other institutional sources and synchronizes it to the system's document and vector repositories. Its most important responsibilities are to extract curriculum information, extract event data from email content, and store all this data in a structured format in the system. Input sources include course files, student emails, university website content and APIs as needed, while outputs include structured data, extracted events and updated repositories. This component has dependencies on components such as university systems, email servers, document and vector repositories, and embedding services. Secure data management, high-volume data processing and timely synchronization are the main quality characteristics of this subsystem.

Document Store (SQL + pgvector)

This component, where structured data is stored permanently, is responsible for storing institutional data such as course contents, academic history, user profiles, etc. This system is built on a PostgreSQL database and supports vector-based queries by integrating with the pgvector plugin. Structured data from the data synchronization module is stored here in SQL format and both traditional SQL queries and searches based on vector similarity are supported. High availability is equipped with security measures such as role-based access control and data encryption. Backup and replication mechanisms guarantee system continuity.

Vector Store (ChromaDB)

This component is responsible for storing the vector representations generated through the embedding service and enabling similarity-based search. In particular, it is used to provide access to university documents based on semantic similarity. Text documents are converted into vectors through the embedding service and these vectors, together with the associated metadata, are transferred to the vector repository. The system provides an infrastructure capable of handling high volumes of data, providing fast access and filtering metadata. Backup and replication strategies can be implemented on supported systems.**[2]**

Embedding Service

The Embedding Service is responsible for transforming text data into high-dimensional vector representations, which are essential for semantic similarity search and context enrichment within the Retrieval-Augmented Generation (RAG) pipeline. This service utilizes state-of-the-art pretrained embedding models (such as those from the Sentence-Transformers or Hugging Face model libraries) to generate dense vector encodings from academic documents, user queries, and parsed email content. These embeddings are then forwarded to both the Vector Store for persistent semantic indexing and the RAG Orchestrator for real-time contextual matching. The service is designed to operate with high throughput, supporting concurrent requests and ensuring low-latency embedding generation. To enhance robustness, it includes retry mechanisms and a fallback strategy in case of model or runtime failures. By leveraging pretrained models rather than training embeddings from scratch, the system benefits from reduced computational cost and faster deployment while still achieving high-quality semantic representations.

Retriever & RAG Orchestrator

This component takes questions from users, collects context information from both the document store and the vector store, and prepares a meaningful prompt for the large language model (LLM). Working with the Embedding service, it accesses the vector representation of the user's question, retrieves the most relevant documents and combines them to produce the optimal input data for the LLM. High-performance querying and timely response generation is the primary goal of this component. Measures such as logging and retries are taken in case of errors.

LLM Inference

This component, the central AI module of the system, generates natural language responses using the enriched context provided by Retriever & RAG Orchestrator. Using a pre-fine-tuned language model (fine-tuning of an open-source LLM), the user is presented with accurate, consistent and understandable responses. The service is configured to be capable of handling multiple simultaneous user requests. It has a low latency tolerance and is designed to be flexible enough to switch to a more robust model when needed.

Chat (Multi-lingual) & Organizer

Interacting with the user interface, this component manages chat sessions, offers multilingual support and initiates event organization processes when needed. It receives questions from the user, processes responses from the LLM and displays them to the user. It also provides calendar integration by routing the collected data to the Collector Agent component. Session management and error logging mechanisms are in place in case of errors.

Collector Agent (Email + Calendar Integration)

This agent accesses student e-mail boxes and Google Calendar service with the user's permission. It detects events such as exams, meetings and project submissions in the e-mails and integrates them into the calendar system. Reliable data processing is ensured for accurate and timely processing of events. For security, all connections are encrypted and access is only granted to authorized components.**[3]**

API Gateway & UI

This component, which connects the system to the outside world, manages user authentication, data routing and client interfaces. It communicates with the Chat module to provide responses to the user in a web-based interface. API endpoints handle HTTP requests from the user, enabling interaction with the system. It has been developed taking into account factors that affect the user experience such as high availability, secure session management and fast loading times.

Monitoring, Logging & Alerting

Monitoring the health of all system components and sending notifications in case of errors, this module collects and analyzes performance metrics. This component, which is especially important for system continuity and reliability, detects anomalies in a timely manner and notifies administrators. If necessary, it offers live monitoring via visual dashboards. Daily data and performance metrics can be used in decision-making processes to improve the system.

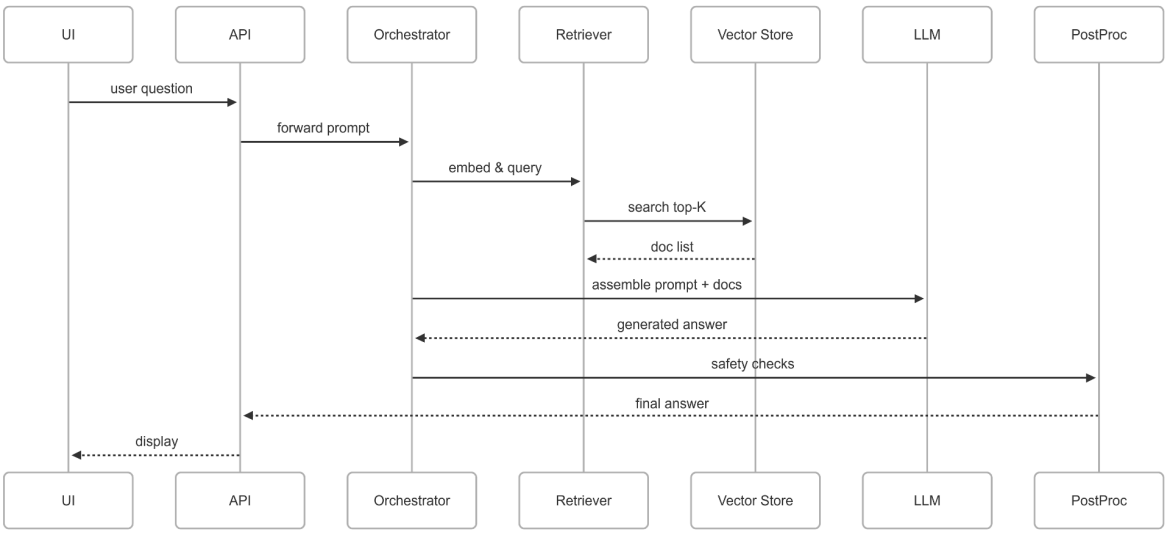

Figure 1 - Sequence Diagram [4]

2.3 Hardware/software mapping

This outlines the proposed hardware and software design required to implement and execute the Project SAGE subsystems. The mapping is aimed at modularity, scalability, security, and economic efficiency so that all software components will be executed on appropriate hardware infrastructure for its performance needs.

a. Hardware Resources

Hybrid deployment is the pattern observed, wherein cloud-scalable infrastructure is employed and in-situ development environments are utilized. The majority of the services will stay in the cloud to enjoy elasticity, managed services, and geo-availability.

GPU Compute Nodes
The GPU compute nodes (if needed) are designed for applications that require high computational performance such as large language model (LLM) inference and likely embedding generation. It is preferable for these nodes to support multi-core processing, be high memory nodes, and have recent GPU units to efficiently execute high-intensity workloads. Fast data access and throughputs supported by SSD-backed storage must be made available. While deployment will utilize GPU-accelerated, scalable virtual machine instances, the final selection will depend on the geographic placement and GPU remaining available. Infrastructures that will support this will have low-latency internal networking in the cloud region, and managed public access through secure endpoints will be made available.

CPU-Optimized Compute Nodes
CPU-optimized (if needed) compute nodes are intended to run compute-bound subsystems that can execute without GPU acceleration, such as a data ingestion mechanism, orchestration, retrieval, and chat processing subsystems. The nodes will have multi-cores CPUs, medium to high RAM, and SSD storage to support responsive and reliable performance. Deployment will make use of scalable virtual machine images intended for general-purpose compute workloads, offering the flexibility to allow users to scale their solution up or down by the demand wanting to be met. Low-latency internal networking is a requirement to ensure various co-hosted services, e.g. the databases and embedded components, can all communicate quickly to improve the overall response time and keep all components properly integrated.

Managed Databases
Managed databases will accommodate structured and vector data storage needs. For structured data, a relational database service (like PostgreSQL) will handle metadata such as user data, document indices, and extracted entities. High-dimensional vector embeddings for semantic retrieval will be stored in a dedicated vector database, ChromaDB. Both database services will be designed for vertical and horizontal scaling to handle increased data demands while maintaining performance and reliability. Access to the databases will be limited to secure internal networks, with adequate access controls in place to protect the data and ensure security compliance.

Container Image Registry
The project development process will primarily take place in a local environment. All backend services will be containerized and run on developer machines using Docker Compose-like tools. This approach will provide flexibility in the development and testing processes, while minimizing compatibility issues in the transition to the production environment. In addition, mock versions of third-party services can be used during the testing phase.

In the later stages, the project will be moved to a cloud platform if needed to increase the scalability of the system or to manage the user load. At this point, hosting on cloud service providers such as AWS (Amazon Web Services) or DigitalOcean is planned. In the cloud environment, all containers will be pulled from a centralized image repository and the services will be managed with the help of an orchestration platform (e.g. Kubernetes) for automatic scaling, load balancing and high availability.

b. Software Deployment
Software components are designed to be modular and deployable individually as containerized services, typically as containerized services. They are mapped to hardware according to their computational and I/O requirements:

LLM Inference Service (GPU)
The LLM inference service is deployed on GPU compute nodes within containerized environments, enabling efficient execution of large language model queries. It operates under the control of either manual triggers or an automatic scheduler, depending on the workload and orchestration strategy. The service communicates with upstream components such as retrievers and orchestrators through well-defined APIs, facilitating seamless integration and coordination within the overall system architecture. This setup ensures scalable, high-performance inference while maintaining flexibility in service management.

Retriever & Orchestration Services
Retriever and orchestration services are running on compute nodes optimized to respond to embedding queries, perform similarity searches, and call LLMs. They internally talk to the embedding service via an API, the vector database (ChromaDB), and an inference endpoint for the LLM, allowing for the efficient, seamless, and secure operation of the multiple components.

Chat and User Interaction Handler
The chat and user interaction handler handles user sessions and interfaces with the orchestration backend to respond to requests. It interfaces externally via the API Gateway and communicates internally with other services via APIs to maintain proper and secure interaction flow.

Collector Agent
The collector agent is either an event-driven or scheduled component that extracts information from third-party services (e.g., email, calendars, etc.), processes the data, and stores it for later use. It handles secure access to data using API tokens and encrypted transport. Data is processed in memory and deleted immediately after ingestion to minimize data retention and security risks.

Monitoring, Logging, and Alerting Agents
Components of monitoring, logging, and alerting operate as background processes or service integrations on each node. These components monitor performance metrics, system health, and error rates. These processes trigger alerts when metrics exceed thresholds you've defined.

Database Access
Database access consists of both structured and vector stores. The structured store is where the processed documents are cached. It also serves as the centralized metadata store, coordinating all session records for users. The vector store is used for quick similarity searches during the retrieval process. The vector store will be accessed through specialized APIs or libraries associated with the vector store for semantic query access.

API Gateway and User Interface
The API Gateway is the authorized entry point for backend services. It handles routing, throttles user requests for each service, performs authentication, and usually performs additional checks on received request types. The UI consists of static frontend assets hosted on a content delivery system and only communicates with the backend using secure HTTPS requests.

c. Scalability & Load Distribution
The architecture will support horizontal and vertical scaling to ensure responsiveness in variable load conditions.

Colocation and Latency Minimization
To reduce latency, we will co-deploy high-frequency communication components, such as the retriever and the embedding service, in order to limit inter-node traffic. Similarly, any latency-sensitive physical service such as inference and vector search, will be hosted in the nearest network proximity to provide high response time and to reduce latency.

Operational Considerations
Operational implications call out some of the primary sources of performance bottlenecks: LLM inference is GPU-bound, dependent on parallelization and model size; embedding and similarity search are memory-bound with large amounts of RAM and fast access to vector indices; and data ingestion is CPU- and I/O-bound, requiring fast parsing and preprocessing to ensure throughput.

Security

Security controls will implement least privilege on all components. PII will only be stored when absolutely necessary, and will be securely deleted once the processing of said PII is complete. Full access control and audit logging will be established on all services to provide accountability for access to data, and to protect sensitive information.

Cost Management

Capacity will be allocated efficiently, reserving higher-cost resources for critical tasks like inference, while using lower-cost compute for preprocessing and retrieval. Regular evaluations of cost-benefit trade-offs will guide resource allocation to optimize overall expenses.

During runtime, a user initiates a request through the Chat UI. The Retriever & Orchestrator processes the query, gathering context from the academic database and pre-parsed event data from the Email & Calendar Collector. It then leverages the Embedding Service and ChromaDB for semantic retrieval, prepares the prompt, and forwards it to the LLM Inference module. The resulting response is returned to the Chat UI for display. This flow enables personalized, dynamic academic assistance for TEDU students.

Figure 2 - SAGE Data Flow Diagram

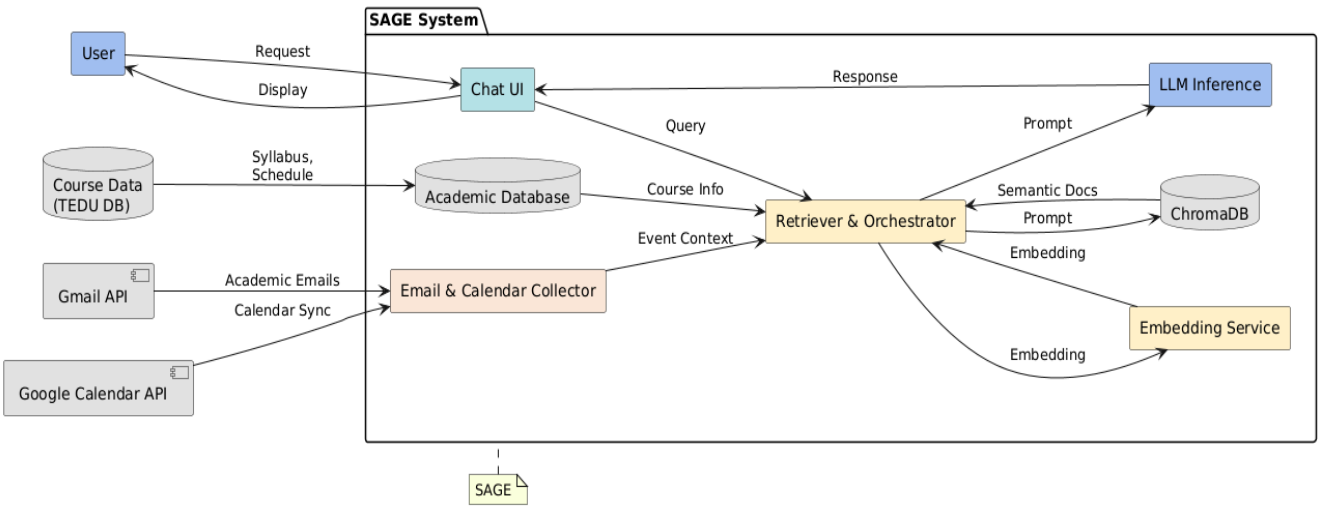

Figure 2 - Data Flow Diagram [5]

2.4 Persistent data management

The persistent data management strategy for Project SAGE has been designed to ensure long-term storage, consistency, accessibility, and integrity of both structured and unstructured data required to deliver the system's full functionality. The architecture relies on a hybrid data persistence approach, using both relational and vector databases to accommodate the needs of semantic retrieval and traditional querying.

Structured academic and user-related data including course information, curriculum metadata, user session logs, student preferences, and email-extracted events are stored in a PostgreSQL database enhanced with the pgvector extension. This extension allows vector representations of structured content to be stored alongside traditional tabular data, facilitating hybrid querying mechanisms that combine SQL-based filters with similarity search over embeddings (vectors).

For unstructured and high-dimensional data such as document embeddings used in semantic retrieval, the system utilizes a dedicated Vector Store, implemented using ChromaDB. This store manages document-level embeddings derived from university documents, syllabus, and email content, and supports efficient nearest neighbor search operations with metadata filtering. The Vector Store is continuously updated as new documents are ingested, ensuring that the LLM receives relevant and timely context.**[2]**

Data is ingested, processed, and synchronized into these stores via the Data Ingestion & Synchronization subsystem. All data pipelines are designed to be fault-tolerant and support incremental updates to avoid data duplication. Backup strategies, including scheduled snapshots and replication for the relational database and vector store, ensure resilience against data loss and system failures.

To preserve data privacy and comply with data protection regulations (e.g., KVKK), all persistent data is encrypted at rest, and personally identifiable information (PII) is either anonymized or minimized to the extent necessary for functionality. Temporary data is processed in-memory and discarded post-processing, ensuring minimal long-term exposure of sensitive information.

Together, this persistent data management framework supports scalable and secure storage solutions for Project SAGE’s operation, enabling responsive and intelligent interactions while ensuring academic integrity, data protection, and reliability.

2.5 Access control and security

Access and data security are underlying principles when it comes to the design for SAGE. The system is specifically built to be consistent with data protection legislation like KVKK and GDPR.

Authentication and Authorization: All users are protected by secure login facilities. OAuth 2.0 is utilized for third-party integrations (like Google Calendar). Role-Based Access Control (RBAC) enforces that only legitimate users (like students, developers) have access to respective system modules.

Data Encryption: The communication between frontend, backend, and third-party services is all encrypted through HTTPS. The sensitive data within the database (tokens and email access status, for example) is stored through encryption-at-rest and in-transit security.

Privacy Protections: Explicit consent is required from the students prior to scanning of their inbox. A revocation feature is also included, permitting users to opt out and delete stored personal data. Email scanning and event generation logs are anonymized wherever possible.

Audit logging: All major actions, including data changes, attempts to log in, and third-party API usage, are logged. The logs are frequently checked for anomalies.

2.6 Global software control

SAGE's global control software architecture achieves coordinated behavior and coordinated communication between subsystems throughout, particularly during response generation and task orchestration.

Conversation Orchestration: The central control module is responsible for coordinating user sessions, context, and task delegation (for example, when to call the embedding module or the calendar integration). This enables continuity between dialogue and task flow.

Data Synchronization: Email scanning is done on an interval (e.g., 30-minute intervals) and causes event database and calendar to be updated. A task scheduler prevents overlapping scans and handles failure with retries.

RAG Workflow Coordination: The orchestrator bridges the embedding service, vector store, and LLM inference. It produces embeddings, loads relevant chunks, formulates a prompt, and calls the model when it receives a query. Mechanisms for failover guarantee reliability.

Notification System: Push notifications and alerts (for instance, reminders for an exam) are sent out using the backend scheduler, coordinated with email and calendar events.

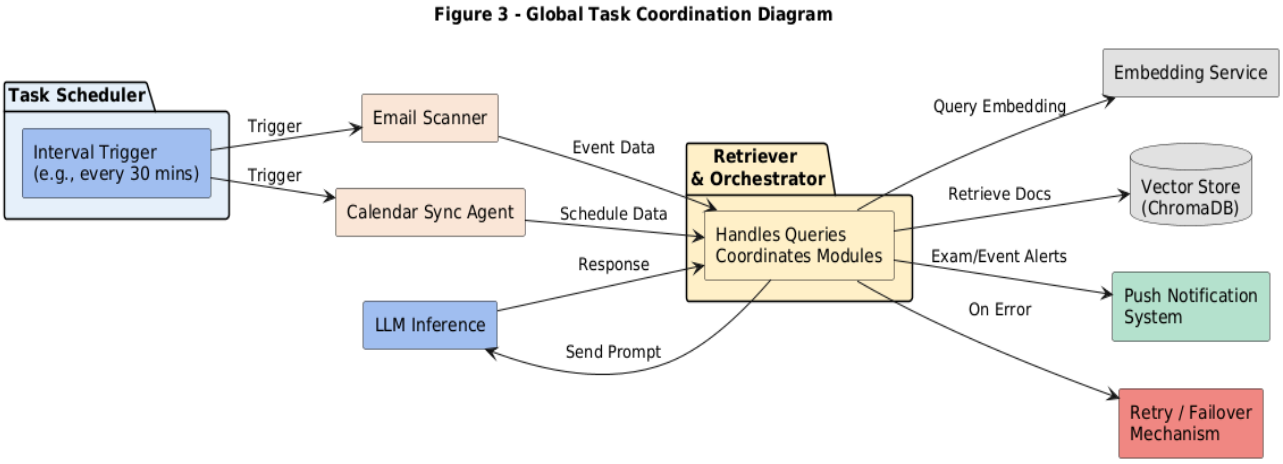

Figure 3 - Global Task Coordination Diagram [5]

2.7 Boundary conditions

Boundary conditions describe the system's behavior for edge situations, ranging from startup/shutdown sequences through failure recovery and degradation operations.

Startup: At system launch, database connectivity and API credentials are checked. The system requests a reauthorization for email/calendar tokens in case they are invalid. Precomputed embeddings are used to warm vector stores and mitigate cold-start latency.

Shutdown: Active sessions are saved during shutdown. In-progress calendar synchronization operations are completed, and pending background jobs are scheduled for retry on restart.

Failure Handling: In the event the embedding service or vector database is down, the system resorts to cached values. On calendar API rate limits, we schedule retries using an exponential backoff.

High Load Management: Auto-scaling rules are invoked during spikes in loads (e.g., registration periods for courses). Queuing mechanisms order chatbot responses by the urgency of the task and place ceilings on concurrent embedding generation requests.

Data Consistency Reconciliation: A reconciliation system makes event data parsed from email consistent with the present calendar, correcting discrepancies when it encounters any.

With these protections, the SAGE system is robust and user-trustworthy, even in uncharacteristic or failure-vulnerable situations.

3. Subsystem services

Project SAGE's architecture consists of multiple independent but integrated subsystems. Each subsystem has a specific functional responsibility and communicates with other components through APIs in a service-based approach. The interaction between these services enables the system as a whole to provide a consistent, secure and user-centered experience.

The Chat & Organizer Service is at the center of user interaction. This service receives student-submitted questions, maintains the context of the conversation and transmits LLM-generated responses to the user in the appropriate format. It also routes calendar-related requests from the user to the Collector Agent service. The Chat service is invoked by API Gateway via REST API and offers multi-language support.

Retriever & RAG Orchestrator Service makes sense of incoming user queries and collects relevant context. It communicates with the Document Store and Vector Store to retrieve the necessary information, and combines them to create a context-augmented prompt for LLM. This service also integrates with the Embedding Service to vectorize incoming queries and select semantically meaningful documents.**[1]**

The LLM Inference Service processes contextual prompts and generates natural language responses. This service runs a big language model that is trained and fine-tuned with institutional data. Based on input from the RAG Orchestrator, it returns descriptive responses to the user about academic, administrative and social issues. The service will be local if it is wanted and will be hosted on GPU-enabled infrastructures for performance.

Data Ingestion & Synchronization Service is a critical component that ensures the system's information timeliness and data integrity. It collects data from the university website, course catalogs and student emails, parses, structures and synchronizes this data to the Document and Vector repositories. This service also performs preprocessing tasks and guarantees the continuity of data flow.

The Collector Agent Service accesses student e-mail boxes with user authorization and automatically detects academic events (exams, deadlines, meetings). These detected events are added to the student calendar via Google Calendar API. The service is designed to be fault tolerant and configured to take user feedback into account.

Monitoring & Logging Service collects metrics from all subcomponents of the system, keeps log records and generates alarms in case of exceeding the specified thresholds. This service plays a critical role in debugging and performance optimization processes as well as guaranteeing the healthy operation of the system.

Finally, API Gateway & UI Service is the layer where the system interacts with the outside world. It receives user requests, performs authorization processes and directs them to the relevant services. It also hosts the web-based user interface and provides a user-friendly chatbot experience.

Each of these subsystem services will be developed in accordance with microservice principles and will be run in a distributed environment according to scalability, flexibility and security priorities.

The basic API and function calls of each subsystem with each other can be summarized as follows:

Chat & Organizer Service receives requests from the user and forwards these requests to Retriever & RAG Orchestrator Service via REST API. At the same time, it forwards requests for the user's calendar to the Collector Agent Service. Returns to the user by receiving LLM responses.

The Retriever & RAG Orchestrator Service vectorizes the incoming natural language queries through the Embedding Service (embed(text)) and then performs a similarity-based search on the Vector Store (querySimilarDocuments(embedding)). It also performs SQL-based context queries through the Document Store (fetchStructuredContext()). The retrieved context is sent as a prompt to the LLM Inference Service (generateResponse(prompt)).**[1]**

The LLM Inference Service generates a response using the context-augmented prompt information from the Retriever & RAG Orchestrator (generateLLLMOutput(prompt)) and returns this response to the Chat Service.**[1]**

The Data Ingestion & Synchronization Service extracts data from university web pages and email content (fetchWebData(), fetchEmailData()), processes this data (preprocess()) and sends it to the Document Store as structured data (insertStructuredData()) and to the Embedding Service to create embeddings (createEmbeddings(text)).

The Collector Agent Service pulls event information from email servers (scanForEvents()) and pushes the detected dates to the calendar via the Google Calendar API (pushToCalendar()). It communicates with the Chat Service via REST API and shares event details.**[3]**

Embedding Service converts incoming text data into vectors with the encode() function and passes these vectors to both Vector Store (storeEmbedding()) and RAG Orchestrator (returnEmbedding()).**[1]**

Monitoring & Logging Service collects logs from all services (logEvent()), processes system metrics (recordMetric()), and generates alerts to administrators in case of errors/alarms (triggerAlert()).

The API Gateway & UI Service receives all user requests (receiveUserQuery()), authenticates them (authenticateUser()), and then forwards them to the Chat Service (forwardQueryToChat()).

Thanks to this structure, all services cooperate with each other within the framework of certain roles and ensure the holistic functioning of the system.

4. Glossary

AI (Artificial Intelligence): It enables computer systems to mimic human-like thinking, decision-making and learning processes. Used in Project SAGE for natural language processing and automatic response generation.

API (Application Programming Interface): Application programming interface. It allows system components to communicate with each other. Calls between all microservices are made through these structures.

Chatbot: An artificial intelligence interface that answers students' academic and administrative questions in natural language. It is the first point of contact for users; managed by the Chat & Organizer Service.

Context-Augmented Prompt: It is an input statement in which information obtained from external sources (document store, vector store) is added in accordance with the user's query. It is sent to the LLM Inference component.

Embedding: The transformation of text into a vector representation. This process enables semantic similarity searches. It is performed by the Embedding Service.

Embedding Service: The service that translates the incoming text data into a vector representation. It provides output to both the RAG Orchestrator and the Vector Store.

Google Calendar API: External API that automatically adds exam and meeting information to the student calendar. It is used in an integrated manner by the Collector Agent.

KVKK (Personal Data Protection Law): The law regulating the secure processing of personal data in Turkey. All system components are developed to work in accordance with this legislation.

LangChain: An open source Python framework that facilitates LLMs to work in context with external data sources. Can be used within the RAG flow.

LLM (Large Language Model): A natural language model trained on large datasets. Generates system responses; run through the inference module.

Monitoring: A structure that monitors the status of system components, collects metrics and detects problems. Part of the Monitoring & Logging service.

pgvector: It is a plugin that allows similarity-based queries with vector data on PostgreSQL. It is used in Document Store.

RAG (Retrieval-Augmented Generation): A structure that combines both retrieval and generation techniques in query answering. It forms the basic architecture of the system.

Semantic Search: A search method based on meaning instead of keywords. Embeddings and Vector Store provide this functionality.

Vector Store: The data store where vector representations are stored. Semantic similarity searches are performed here. Implemented with ChromaDB.

Web Scraping: A technique for collecting data from websites. The Data Ingestion service uses this method to extract curriculum information from university pages.

5. References

[1] Lewis, M., Pérez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., … & Riedel, S. (2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. In Advances in Neural Information Processing Systems, 34, 9469–9480. https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[2] ChromaDB. (2024). *Chroma: The Open-Source Embeddable Vector Database*. Retrieved from https://docs.trychroma.com/docs/overview/introduction

[3] Google. (2024). *Google Calendar API Documentation*. Retrieved from - https://developers.google.com/calendar

[4] Lucidchart. *Figure 1 - Sequence Diagram.* Retrieved from – https://www.lucidchart.com

[5] PlantUML. *Figure 2 - Data Flow Diagram, Figure 3 - Global Task Coordination Diagram.* Retrieved from – https://plantuml.com