TED UNIVERSITY

CMPE 492-O SENIOR PROJECT

Low Level Design Report : **Project SAGE**

Arda Taha Sökelen

Mehmet Alp Demiral

Miray Aday

Oğuzhan Altın

1. **Introduction**

Project SAGE (Smart AI Guide for Education) is an intelligent and privacy-focused chatbot system designed to support TED University students' academic and social life. This intelligent chatbot employs modern Artificial Intelligence (AI), Natural Language Processing (NLP), and information retrieval technologies to offer contextually appropriate, personalized support throughout students' university lives. SAGE is designed to be a modular and flexible platform that delivers accurate execution of academic activities, supports time management and calendars through automation, and provides helpful context for common situations encountered socially and academically at TEDU.

While the chatbot is considered as a single system, it contains multiple different modules that are interdependent, each assigned to do a specific function, including searching for academic data, inferring language model results, parsing emails, and synchronizing events. The chatbot will leverage a fine-tuned Large Language Model (LLM), in combination with a Retrieval-Augmented Generation (RAG) pipeline, to ensure all responses are grounded, factually accurate, and contextually aware relative to institutional data. The chatbot will also use the Google Calendar API to automatically detect and schedule important academic events based on information (styles and texting) from the student's email inbox, minimizing reliance on the student's input and time, and to mitigate data entry errors.

The Low-Level Design Report presents a detailed technical understanding of how each component of SAGE works within itself, and works as a collection of components to achieve the goals of the system. This report presents the description of class structures, interfaces, data storage, and control flow on the design of the chatbot. This report aligns with the IEEE standards[1] for Software Design Descriptions, which assures the design is systematic, traceable, and maintainable. This document represents the implementation-level architecture in detail and serves as a perfect implementation guide, linking the high-level conceptual design of the software to the implementation of the application level, with minimal ambiguity to support the future scaling, testing, and deployment of the design.

1.1 Object Design Trade-offs

The design of Project SAGE needed to balance performance, scalability, data privacy, and maintainability. Prioritizing sub-second responses in conversational interactions for all users means that numerous optimization decisions were needed at the object and architectural levels.

Performance versus Accuracy.
The LLM-driven query engine uses a Retrieval-Augmented Generation (RAG) pipeline from the language-modeling AI community when answering queries. To minimize latency, the SAGE project limits the records retrieved to the top-K, often only five semantically relevant documents per query. Although reliability may slightly suffer as a result--meaning less comprehensive contextual accuracy--as a result, the returned answers are delivered back to the user quickly, even with 500 users replying within 1.5 seconds of asking the question.

Privacy versus Accessibility.
To be compliant with KVKK and GDPR, Project SAGE achieves all data preparation and processing about users within a secure environment. A local fine-tuned LLM was considered preferable rather than an external API model from an ethical standpoint, which allows the project to maintain full control of user-generated data, even as it executes all data processing efficiently and sometimes without significant processing overhead.

Usability vs. Security:
Multi Factor Authentication were used to protect access to that data and API communication. Both security mechanisms introduced minimal friction for the user onboarding and added trust and integrity to the system, especially for email scanning.

Freshness vs. Consistency:
System components depend primarily on models involving scheduled synchronization (rather than continuously updating in real-time). This arrangement increases system reliability; reduces rate limit risks with third-party APIs; and builds tolerances for faults. Any eventual consistency ensures that modules will eventually reach the latest version in each synchronization cycle.

Together, these trade-offs create a design that focuses on principles for ethical AI, reliable systems, and user trust while still performing well under low latency conditions.

1.2 Interface Documentation Guidelines

This section explains the common rules and methods used to design and describe both internal and external interfaces in all Project SAGE modules. Using the same design standards across the system makes it easier to maintain, expand, and connect different parts securely and smoothly. All services use RESTful APIs that communicate over HTTPS for safety and use JSON as the standard format for data exchange. These shared rules make the modules work well together, reduce technical complexity, and follow modern software development practices.

1.2.1 General Interface Requirements

All components of Project SAGE comply with a common communication protocol designed to ensure interoperability and consistent performance, and effective security over the entire system. All communications between services, such as the LLM Inference Engine, the RAG Orchestrator, and the Data Synchronization modules, are authenticated using JSON Web Tokens(JWT) signed with encryption to provide directly verifiable, resistant identity authentication. For any external integrations within SAGE, such as Google Calendar and Gmail APIs, the framework puts in place OAuth 2.0 authorization flows that facilitate refresh token rotation automatically to facilitate seamless access while adhering to contemporary standards of authentication.

To ensure the privacy and confidentiality of all data in transit, all network communication channels enforce TLS and encrypted sensitive data in rest uses hashing. The system never retains or transmits personal data in plaintext, and adheres to privacy engineering practices in support of KVKK and GDPR data protection measures.

Error handling and system observability conforms to a structured methodology, facilitating reliability and traceability. Error reporting for each API response must follow the ErrorEnvelope structure defined as {code, message, details?, traceId}, ensuring consistency in error reporting. Errors and anomalies are logged centrally with correlation identifiers (i.e. trace Ids) keeping the ability to trace distributed transactions across microservices and streamline troubleshooting.

The user experience layer of SAGE incorporates localization as a fundamental part of its functionality. In total, every text-based response and interface feature recognizes the Accept-Language HTTP header and its values to render outputs in real-time accordingly, either English or Turkish, as to respect the bilingual nature of academic life at TED University.

Lastly, in addition to setting the stage for service evolution, the software environment allows for backward compatibility and is supported by a formalized semantic versioning scheme. Additionally, minor features added and none-breaking changes are designated by updating the MINOR version number, and whenever a change is made that introduces a change to an existing interface, there is an update to the MAJOR version. This semantic way of tracking version numbers creates an understandable method for tracking version updates, reliable and consistent upgrades, and support of long-term maintenance of the software ecosystem.

Together, the communication and interoperability standards described (and any future modifications) provide a robust and tacitly extensible and resilient foundation for Project SAGE by promoting modular growth, safety in data exchange, and consistent behavior across all software environments where it is operational.

1.2.2 Internal System Interfaces

The Project SAGE internal system interfaces define the architecture of communication pathways for microservices to share data and coordinate actions for common functionality. Each communication is built on consistent APIs, data contracts, and security policies to ensure consistency, traceability, and interoperability across the platform.

The Chat & Organizer module communicates directly with the RAG Orchestrator via the /rag/answer endpoint, sending a structured request of a RagQuery object including the userId, question, locale, and sessionId. Upon receiving the RagQuery, the Orchestrator retrieves the appropriate contextual information, processes it using the retrieval-augmented pipeline, and sends back a RagAnswer response with the generated text, citations, the identifier of the model, and the latency in milliseconds. This interface supports real-time and contextually reliable talking between the chatbot front-end and the core reasoning engine.

To facilitate both semantic search and knowledge retrieval, the RAG Orchestrator communicates with the Embedding Service and Vector Store. The Embedding Service exposes an /embed endpoint that accepts raw text and converts the text to high-dimensional vector representations. The vectors are then passed to the Vector Store through the /vectors/search endpoint, which runs the semantic similarity query against the ChromaDB repo. The ranked retrieval results are then returned to the Orchestrator which allows for the assembly of a context-enriched prompt for the LLM inference.

The Collector Agent acts as a bridge between SAGE’s internal infrastructure (such as the Embedding Service and Vector Store) and the data sources. The Agent connects to Gmail API (via /messages/list) in order to retrieve any email containing academic delimiters such as “exam,” “assignment,” or “meeting.” Then the extracted event information is updated with the Google Calendar API to the students personal calendar via the /events/insert endpoint.

Operational integrity is maintained through the Monitoring System, which continuously aggregates performance metrics, log entries, and error statistics from all services. When thresholds for latency, error rate, or resource utilization are exceeded, the monitoring subsystem automatically triggers alerts for administrative evaluation, allowing rapid identification and resolution of system anomalies.

All internal communications are governed by strict input validation, authentication, and rate-limiting mechanisms to prevent unauthorized access and ensure stable system performance. This cohesive interface design guarantees that data flows within Project SAGE are secure, efficient, and fully synchronized across all microservices.

1.2.3 Data Formats and Standards

Project SAGE enforces uniform data standards across all subsystems to maintain consistency, interoperability, and precision in data exchange. All entities within the system are assigned Universally Unique Lexicographically Sortable Identifiers, ensuring globally unique and chronologically ordered record identification. Temporary data strictly adheres to the ISO-8601 timestamp standard with timezone precision set to Europe/Istanbul UTC +3 timezone, ensuring synchronization between distributed services and external APIs. All text is encoded using UTF-8 to support multilingual data, prevent corruption, and guarantee compatibility between each system.

Database schemas follow a unified naming convention that uses lowercase letters and underscores for clarity and consistency (e.g., course_code, event_date). Academic data such as course descriptions, credit structures, and instructors is stored and exchanged in structured JSON format to promote interoperability between components. The following example represents a standardized Event Schema used in both the Document Store and Vector Store layers:

```
{
  "code": "CMPE 114",
  "name": "Fundamentals of Programming II",
  "hours": "2+0+2",
  "credits": "3",
  "ects": "6",
  "semester": "Fall",
  "sections": "01, 02, 03, 04",
  "instructors": "Eren Ulu, Tansel Dökeroğlu",
  "syllabus_url": "https://cmpe.tedu.edu.tr/en/izlence-detail/50296028",
  "course_code": "CMPE 114",
  "course_title": "Fundamentals of Programming II",
  "level": "BS",
  "credit_hours": "(2+0+2) 3 TEDU Credits, 6 ECTS Credits",
  "academic_year": "2025",
  "catalog_description": "Classes. Objects. Operator overloading. Packaging. Linked
lists. Queues. Stacks. Searching and sorting algorithms.",
  "prerequisites": [
    "CMPE 112",
    "CMPE 113"
  ],
  "corequisites": [],
  "ins
tructor": "Eren Ulu"}
```

This schema ensures structured and machine-readable representation of course and event information, enabling reliable data ingestion, querying, and synchronization across services. Vector embeddings derived from these records are stored in ChromaDB[8], indexed with metadata for semantic similarity queries. Additionally, PostgreSQL's pgvector extension allows hybrid SQL + vector queries, combining structured academic data retrieval with semantic context matching resulting in faster, more accurate, and context-aware responses within the SAGE ecosystem.

1.3 **Engineering Standards**

To ensure safety, maintainability and professional responsibility, our project Sage sticks with established engineering and ethical standards. With the help of these frameworks we enhance every stage of the system's design, development, and validation processes.

- IEEE 1016: This standard defines the structural framework and documentation conventions followed throughout this report to ensure clarity and integrity in software design representation.[1]

- IEEE 830: This standard maintains traceability between design elements and system requirements to ensure conformance to the prior specification documents and consistency in design.[2]

- ISO/IEC 25010: The standard establishes measurable quality criteria for performance, reliability, usability, and maintainability to provide a basis for design evaluation and quality assurance.[3]
- OWASP ASVS 4.0: This specification implies the best practices for secure API development, authentication mechanisms, and data protection controls across all system modules.[4]

- KVKK & GDPR Compliance: This specification implies that all personal data are processed following Turkish and EU data protection regulations[6], including explicit consent, anonymization, and the right to data erasure.[5]

The Project SAGE system guarantees an organized approach to software development by following these engineering standards, which improves scalability, interoperability, and adherence to industry best practices.

1.4 Definitions, Acronyms and Abbreviations

Acronym	Definition
AI	Artificial Intelligence: the emulation of human cognitive functions by machines, especially computer systems.
RAG	Retrieval-Augmented Generation: A composite framework that consists of document retrieval and generative modeling applied to a content-based context.
LLM	Large Language Model: A type of neural network trained on large amounts of text to complete natural language understanding and generation tasks.
JWT	JSON Web Token: A small, URL-safe way of securely transmitting claims between two parties for purposes of authentication and authorization.
DAO	Data Access Object: A design pattern that provides an abstract interface to data stored using a persistence framework.
pgvector	PostgreSQL Vector Extension: An extension for PostgreSQL that allows for efficient ingestion and similarity search of vector embeddings.[7]
KVKK	The Turkish Personal Data Protection Law The legislation in Türkiye, which regulates the protection and processing of personal data.[6]
GDPR	GDPR: The Regulation in EU Law on data protection and privacy for individuals in the European Union and the European Economic Area.[5]
ULID	Universally Unique Lexicographically Sortable Identifier: A unique identifier that combines a timestamp ordering by design to create randomness.
RBAC	Role-Based Access Control: A security approach that restricts system access to authorized users based on predefined roles.

2. **Packages**

The Project SAGE system architecture is constructed as modular software packages that each accomplish a defined set of functions in the system. This modular approach allows for separation of concerns and the ability to develop, test, and deploy components independently. Each package presents interfaces for communication with other packages, enabling scalable, maintainable, and integrated development.

2.1 **Packages Overview**

Project SAGE has been conceived as a modular system of discrete packages, each with a precise purpose and functionality within the context of overall system functionality. The only interaction among the packages occurs through intentional APIs, which solidly separates out the responsibility of the package, allows for product life cycle maintenance, and makes it clear how they should communicate with each other. The individual, modular, defined-use design of the package allows for independent development, testing, and deployment, which aligns with the modern architectural orientation towards microservices.

The Chat and Organizer Package handles the user interaction layer of the overall system, including session control, intent detection, and response delivery, including in multiple languages. It serves as the primary interface through which the user and the backend intelligence modules connect. The overall package maintains conversational context and identifies and sends user queries to the RAG pipeline and the downstream modules.

The RAG Orchestrator Package is at the center of the overall system. It represents the intelligence of the system. It orchestrates the retrieval and generation workflow of the overall system. It is responsible for connecting the Document Store and the Vector Store, creating prompts that include context when appropriate, and prompts the Large Language Model to generate a response.

In support of this primary function, the Embedding Service Package is responsible for converting input text into high-dimensional vector-based encodings to provide semantic meaning throughout the entire platform. The Vector Store Package, implemented as ChromaDB[8], uses these audit models in order to perform similarity searches in an embedding space, and return the closest contextual data relevant to the user. Additionally, the Document Store Package, which is powered by PostgreSQL with the pgvector extension, maintains structured academic documents including course information, user metadata, and syllabi.

The purpose of the Collector Agent Package is to serve as the integration layer between Project SAGE and external services. It securely connects to institutional email services using the Gmail API to determine such academic events as exams, assignments, or meetings, and syncs it to the student calendar using the Google Agenda API.

Lastly, to provide system reliability, reliability, and transparency, the Monitoring Package executes a continuous collection of operational metrics, errors, and performance data across all the microservices and provides administrators with real-time observability and alerting. Finally, the Web Interface Package provides an accessible, bilingual (English and Turkish) frontend to facilitate usability across platforms and conform to accessibility standards in order to be inclusive.

Each package is completely containerized and deployed independently, allowing for horizontal scalability, rapid fault tolerance, and easier maintenance. This modular packaging creates the potential for Project SAGE to adapt to future growth and upgrades in the technology stack while maintaining a cohesive model of communication and data management.

2.2 **Package Dependencies**

The dependency structure of Project SAGE uses a layered and modular design to reduce coupling and improve system reliability. The User Interface Package has a dependency on the Chat and Organizer Package, the latter of which controls the flow of dialogue and the routing of user requests to back-end services. The Chat Package communicates directly with RAG Orchestrator as the intelligent handler of query requests (e.g., pulling in external data) and with Collector Agent to trigger events for synchronization tasks.

The RAG Orchestrator Package depends on three main subsystems: the Embedding Service for vectorization of data, a **Vector Store (ChromaDB)**[8] for semantic searching, and a **Document Store (PostgreSQL + pgvector)**[7] for structured academic data. The Collector Agent functions in conjunction with external dependencies, and it communicates securely with Gmail as the email parsing mechanism and Google Calendar for event creation in an automated enrich routine. For collecting and inspecting logs, metrics, and performance data across the subsystems the Monitoring Package exists as a high level layer.

This diagram creates a clear model showing how data flows, allows each subsystem to scale and test on its own, and helps prevent issues from affecting other components. Overall, this boosts the system's stability and makes it easier to maintain.

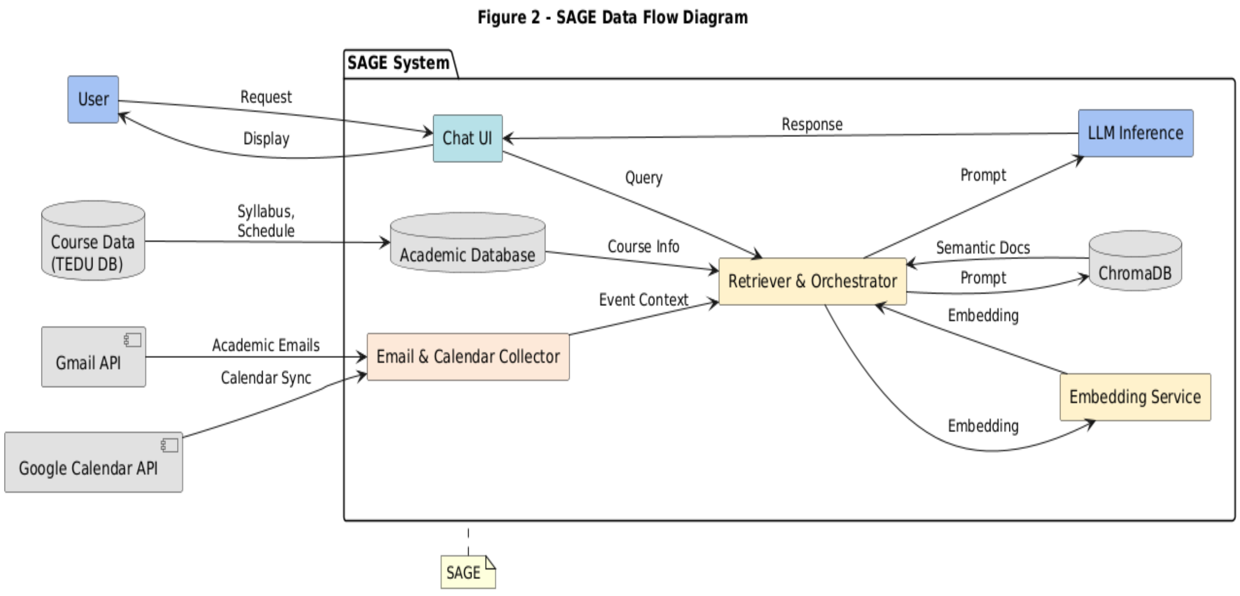

Figure 1. Data Flow Diagram

3. Class Interfaces

3.1 LLMInferenceEngine Class Interface

The LLMInferenceEngine class is the computational heart of Project SAGE, responsible for all natural-language understanding and generation processes. It does this by acting as the interface between the RAG Orchestrator and the locally hosted fine-tuned large language model (LLM) to execute inference operations and thus produce semantically accurate, contextually grounded responses.

Purpose and Role:

This class takes structured prompt objects as input, containing the user's question and relevant contextual information retrieved by the RAG pipeline. It converts these into model-compatible representations, invokes the LLM, and returns normalized, post-processed results. Its behavior directly influences every system response in terms of fluency, reliability, and factual correctness.

Functional Overview:

1. Prompt Processing: It validates and enriches incoming prompts with contextual data, token-length controls, and localized linguistic adjustments.
2. Model Execution Control: Manages the lifecycle of inference sessions and their resource allocation, taking into consideration the execution scheduling across CPU/GPU environments.
3. Normalization: This processes the raw LLM output to remove artifacts, add citation markers, and make it more readable.
4. Performance Monitoring: Records latency, token usage, and throughput metrics, sending them to the Monitoring System for observability.
5. Error Recovery: This implements retry and fallback routines when inference fails, ensuring system stability.
6. Security and Compliance: Anonymizes all user identifiers before processing and guarantees that no sensitive data is retained post-inference.

Interactions:

Takes in prompts from the RAG Orchestrator.

Returns structured responses to the ChatSessionManager for presentation.

Publishes performance metrics to the Monitoring Package.

The LLMInferenceEngine thereby concretizes the reasoning capability of Project SAGE, striking a judicious balance between computational efficiency, ethical constraints, and linguistic coherence within a single abstraction class.

3.2 **ChatSessionManager Class Interface**

The ChatSessionManager class is responsible for user interactions and the persistence of the conversation. It acts as a “glue” that connects users to the system's backend via the user interface, providing unstructured outputs on request inputs and structured requests on unstructured outputs.

Purpose and Role:

This class provides a seamless, secure, and contextually coherent dialogue between the user and the intelligent system. It manages data flow between multiple backend services while tracking conversation history, language preferences, and session metadata.

Functional Overview:

1. It initiates, tracks, and terminates chat sessions using session IDs without the need to store personal information.
2. The intent recognition and classification process determines whether a message is a command, request, or query. It routes the message to the appropriate system and subsystem.
3. Context Gathering is used to improve the LLM's response quality. It collects previous messages, calendar data, and academic information.
4. Localization Management: Uses session metadata. Displays all text content in English or Turkish according to the user's preferred language.
5. Secure Communication: After authentication via JWT tokens, all outgoing requests are transmitted over encrypted HTTPS channels.
6. Error and Status Management: Transitions between idle, loading, responding, and error states are critical for providing clear feedback to the user.
7. User Feedback Loop: Records user responses such as likes or dislikes for adaptive fine-tuning and model improvement.

Interactions: It forwards user requests to the RAG Orchestrator using standard REST API endpoints. After receiving responses from the LLMInferenceEngine, it formats model responses for the user interface. When appropriate, it synchronizes with the DataRepository to retrieve or update academic data. ChatSessionManager encapsulates these techniques to ensure that every conversation remains secure, consistent, and within TED University's guidelines and standards. Furthermore, its modular design will enable future AI assistant integrations, as well as web and mobile platforms, to operate with the same logic without requiring any changes to the architecture.

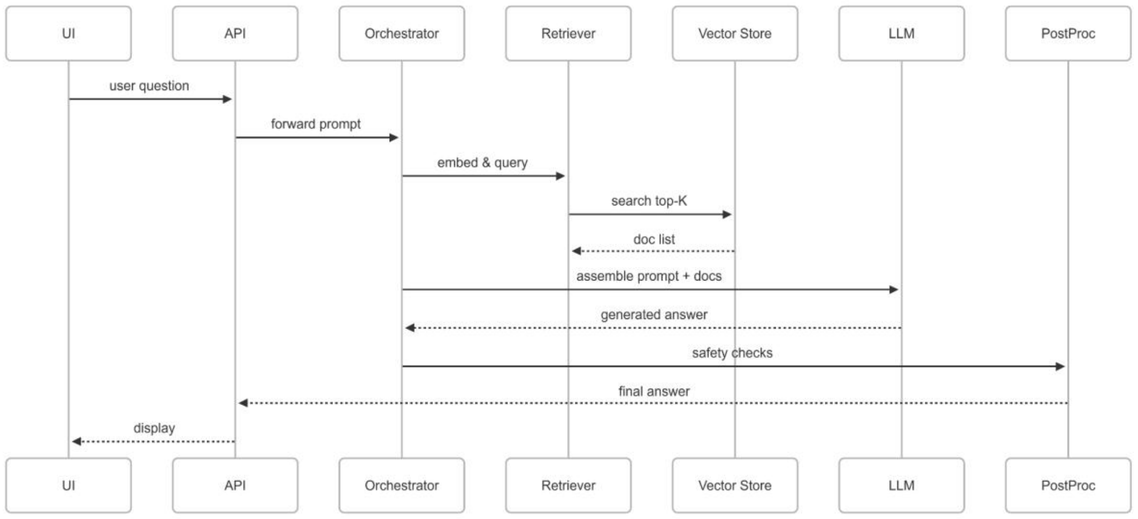

Figure 2. Sequence Diagram

3.3 **DataRepository Class Interface**

The DataRepository class represents the persistent data access layer of Project SAGE. It abstracts all interactions with structured and unstructured data sources, thus providing a uniform interface by means of which higher-level modules can retrieve, store, and manage academic information along with embeddings and user-related metadata.

Purpose and Role:

This class ensures that all operations on data in the system are consistent, traceable, and adhere to privacy regulations. By encapsulating the database logic, it shields the rest of the application from low-level details about the PostgreSQL and ChromaDB queries, promoting maintainability and scalability in the long run.

Functional Overview:

1. Unified Access Layer: Provides standardized methods for performing CRUD operations on relational and vector databases.
2. Schema Integrity: Ensures consistency between relational schemas (courses, events, users) and their respective embeddings in the vector store.
3. Hybrid Query Support: Combines structured SQL queries with semantic vector search to deliver contextually rich results for RAG operations.
4. Data Synchronization: Coordinates updates between the Collector Agent, outside APIs (Gmail, Calendar), and internal databases using scheduled routines.
5. Safety of Transactions: Uses rollback mechanisms and atomic operations to avoid partial updates or orphaned records.
6. Performance Optimization: Uses indexing, caching, and batched writes to maintain low-latency responses under heavy load.
7. Access Control and Encryption: Provides role-based access policies for sensitive data using RBAC, and field-level encryption based on AES.
8. Compliance and Anonymization: Supports KVKK and GDPR requirements by offering anonymization and right-to-erasure routines.

Interactions:

Interface to PostgreSQL for structured data, via pgvector.

Connects to ChromaDB for vector embeddings and similarity search. Communicates with external APIs for event synchronization via the Collector Agent. Provides the LLMInferenceEngine with verified data during contextual retrieval phases. This DataRepository class is the backbone of SAGE's data integrity. It serves to transform scattered academic and contextual data into a well-organized query-ready knowledge base for the AI components to reason over. With the enforcement of strict security and consistency rules, it secures every system output to be based on credible information.

4. **Glossary**

AI (Artificial Intelligence): A part of computer science that is involved in creating machines that can think and learn as people. These models allow them to make decisions, solve problems, and get better at what they do over time.

RAG (Retrieval-Augmented Generation): an AI method that includes both retrieving existing real information and generating texts to produce more accurate and reliable answers than those generated only in one way.

LLM (Large Language Model): A type of computer model trained on a large amount of text to know language and write similar to how people speak and write it.

DAO (Data Access Object): A design approach that makes the system easier to maintain and update by separating the program's use of data from its storage.

ChromaDB (Vector Database): To store and search data using numbers, or vectors, making it simple to locate related terms or subjects.

pgvector (PostgreSQL Vector Extension): Helps the storage and search of vector data, which is helpful for AI functions like identifying related words or texts.[7]

OAuth 2.0 (Open Authorization Framework): It is a security mechanism that permits apps to securely access your data without revealing your password, such as when you log in with Facebook or Google.

KVKK (Kişisel Verilerin Korunması Kanunu): The Turkish Personal Data Protection Law The legislation in Türkiye, which regulates the protection and processing of personal data.

ULID (Universally Unique Lexicographically Sortable Identifier): A unique ID format that also keeps order by time, making it easier for computers to organize and track information.

5. References

[1] IEEE Computer Society, *IEEE Std 1016-2009 – IEEE Standard for Information Technology – Systems Design – Software Design Descriptions*, IEEE, 2009.

[2] IEEE Computer Society, *IEEE Std 830-1998 – IEEE Recommended Practice for Software Requirements Specifications*, IEEE, 1998.

[3] International Organization for Standardization (ISO), *ISO/IEC 25010:2011 – Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models*, ISO, 2011.

[4] OWASP Foundation, *OWASP Application Security Verification Standard (ASVS) 4.0.3*, The OWASP Foundation, 2021. Available: https://owasp.org/ASVS/

[5] European Parliament and Council of the European Union, *Regulation (EU) 2016/679 – General Data Protection Regulation (GDPR)*, Official Journal of the European Union, 2016.

[6] Republic of Türkiye, *Law No. 6698 on the Protection of Personal Data (KVKK)*, Official Gazette of the Republic of Türkiye, 2016.

[7] PostgreSQL Global Development Group, *pgvector – Open Source Vector Similarity Search for PostgreSQL*. Available: https://github.com/pgvector/pgvector

[8] Chroma, *ChromaDB – Open Source Embedding Database*, Chroma, 2024. Available: https://www.trychroma.com/