Spring 2025-2026

CS492 - Senior Design Project

Design Project Final Report

T2526

Ahmed Hatem Haikal - 22001482

Amirhossein Ahani - 22101535

İrfan Hakan Karakoç - 22003421

Mehmet Hakan Yavuz - 22002119

Türker Köken - 22102331

Supervisor: Anıl Koyuncu

Innovation Expert: Haluk Altunel

Contents

1. Introduction

1.1 Purpose of the System

Consensia is a software system developed to support **multi-perspective reasoning** in software engineering and related decision-making tasks. Its main purpose is to enable users to create or select AI personas representing different professional roles, gather their independent responses to a problem, and combine these responses into a final consensus through a Judge LLM. Recent advances in **large language models (LLMs)** have enabled systems capable of reasoning, summarization, and decision-support across multiple domains [1].

Unlike traditional single-agent AI systems, Consensia focuses on **diversity of viewpoints, structured evaluation, and explainable recommendations**. Multi-agent reasoning approaches using language models have been explored in recent research to improve answer quality and reliability through collaborative or adversarial reasoning among agents [2].

In addition to being an interactive application, Consensia also serves as a platform for experimenting with **multi-agent LLM behavior, persona-based reasoning, and explainable AI**. Research on LLM evaluation and reasoning frameworks shows that structured comparison between multiple model outputs can improve interpretability and analysis of generated responses [3].

Overall, Consensia is intended as a **decision-support and analysis tool**, where AI-generated outputs assist human judgment rather than replacing it.

1.2 Design Goals

The design of Consensia is based on both technical and user-oriented goals identified during the specification and analysis phases.

Usability:
The system should provide a clear and intuitive interface that allows users to create personas, submit questions, and review results easily. User-centered interface design principles emphasize clarity and transparency when presenting AI-generated information [4].

Performance:
Since the platform manages multiple LLM interactions in one session, it should provide acceptable response times and efficient handling of persona generation, task processing, and judge evaluation. Efficient orchestration of multiple AI agents has been studied in recent multi-agent AI frameworks [5].

Reliability:
The system should operate consistently under normal conditions and continue functioning even if some external API calls are delayed. Robust system architecture and fault-tolerant service design are essential when integrating external AI services [6].

Maintainability and Modularity:
The system should be designed in a modular way so that core parts such as persona management, task handling, evaluation, and storage can be developed, tested, and improved independently. Modular software architecture is widely recommended in modern software engineering for scalability and maintainability [7].

Scalability and Extendibility:
The architecture should support future growth, including more personas, more complex debate flows, and integration with additional LLM providers or new persona-generation methods.

Security and Responsible Operation:
Since the platform may process user-uploaded CVs and external data, it should protect user information, ensure secure API communication, and support responsible use of generated outputs.

Research Value and Explainability:
The system should also serve as a platform for studying multi-agent LLM behavior. For this reason, it should preserve intermediate outputs and present the reasoning process clearly so that users can understand how the final consensus is formed.

1.3 Definitions, Acronyms, and Abbreviations

This section defines the main terms used in the report.

Persona: An AI-defined role representing a specific professional perspective.

Judge LLM: The model responsible for evaluating persona responses and producing the final consensus.

Consensus: The final recommendation generated after comparing multiple persona answers.

Ground Truth: A trusted or labeled answer used to evaluate system output.

1.4 Overview

Consensia is designed as a layered system consisting of **frontend, backend, API integration, and storage components**. The user interacts with the system through the frontend to create personas, submit questions, and review results. The backend manages the workflow by generating persona responses, sending them to the Judge LLM, and storing the outputs.

The system supports the full process of persona creation, independent response generation, judge-based evaluation, explanation generation, and optional comparison with ground truth. Layered and service-based architectures are commonly used in AI-driven systems to separate user interaction, application logic, and external services [7].

Overall, Consensia functions both as an **end-user application and as a research platform** for exploring multi-agent reasoning, consensus generation, and explainable AI.

2. Current Software Architecture

2.1 Current Software Architecture

The current software architecture of Consensia is based on a **modular multi-agent structure** that supports persona-based reasoning and consensus generation. When a user submits a prompt, the system sends the same input to multiple AI personas, each representing a different perspective. These personas generate separate responses, which are then passed to a **Judge LLM** that compares the outputs, identifies the most useful points, resolves differences, and produces a final consensus response. Multi-agent reasoning using large language models has recently been explored as a way to improve reasoning quality and reliability by combining multiple perspectives [2].

This architecture allows Consensia to move beyond single-answer generation and provide more balanced and explainable results. The system is organized into separate components such as user interface, backend control, persona management, response generation, judge evaluation, and result display. Such **modular architectures** are widely recommended in modern software engineering because they improve maintainability, scalability, and system extensibility [7].

The modular structure makes the architecture clear, easier to manage, and suitable for future expansion. New personas, improved prompting strategies, or different judging methods can be added without changing the whole system. Overall, the current software architecture of Consensia provides a clear flow from user input to final consensus output while preserving flexibility and modularity.

2.2 Competitors

Several existing systems and frameworks relate to the ideas implemented in Consensia. Multi-agent AI frameworks such as AutoGen and CrewAI allow developers to coordinate multiple language model agents and assign them different roles within a task workflow **[5], [8]**. These frameworks provide general infrastructures for building agent-based applications and support role-based interactions between agents.

Another related concept is the use of “LLM-as-a-judge,” where one language model evaluates or ranks the outputs generated by other models. Evaluation frameworks such as MT-Bench and AlpacaEval apply this idea to compare model outputs and assess response quality **[3], [9]**.

However, these systems mainly focus either on agent orchestration or model evaluation. In contrast, Consensia combines persona-based reasoning with judge-based synthesis to produce a final consensus from multiple expert-like perspectives. Personas in Consensia can be generated from user-defined descriptions, CVs, or researcher information collected from academic sources, allowing the system to simulate different professional viewpoints when analyzing a task.

3. Proposed Software Architecture

3.1 Overview

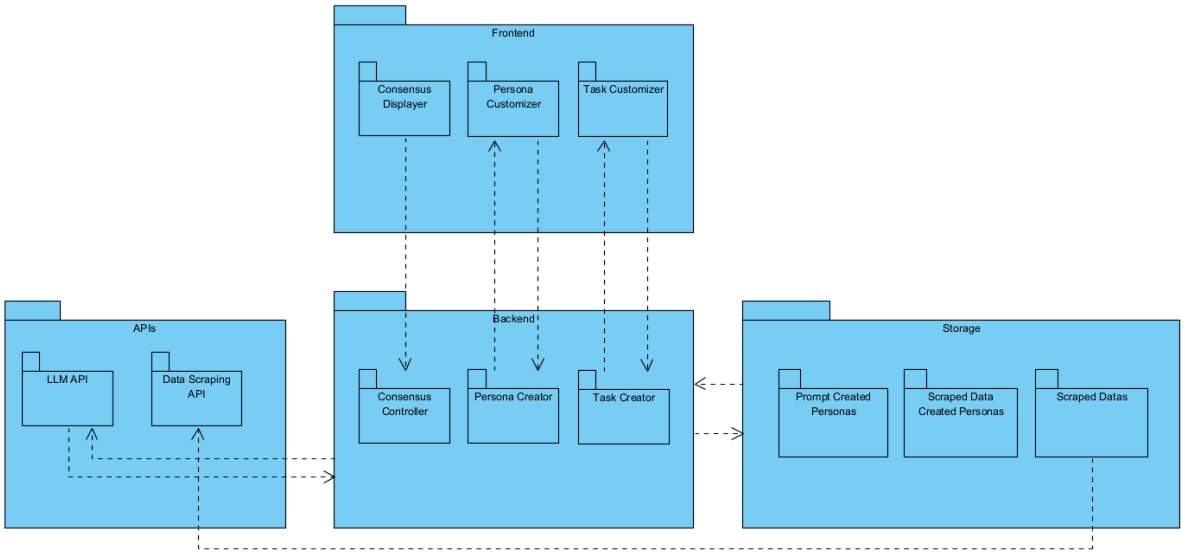
The proposed architecture of Consensia follows a **modular client–server and layered design** consisting of a frontend, backend, API integration layer, and storage layer. This separation allows user interaction, system logic, external services, and data management to be handled independently. Layered architectures are widely used in modern software systems because they improve maintainability, scalability, and separation of concerns [7].

The frontend provides the user interface where users can write questions, select personas, and view responses and the final consensus. The backend manages the core business logic, including persona generation, coordination of agent responses, and interaction with LLM services through APIs.

The storage layer uses a **PostgreSQL database** to store user accounts, sessions, and system data, ensuring that each user's information and preferences are kept separate. PostgreSQL is a widely used open-source relational database system known for reliability and strong data integrity features [10].

Overall, this architecture supports a clear workflow from user input to consensus generation while maintaining modularity and scalability.

3.2 Subsystem Decomposition



The Consensia system is divided into four main subsystems: **frontend, backend, API layer, and storage layer**. This decomposition separates user interaction, system logic, external services, and data management. Subsystem decomposition is a common architectural strategy used to improve modularity and simplify system development [7].

The frontend subsystem includes components such as the **Persona Customizer, Task Customizer, and Consensus Displayer**, which allow users to create personas, define tasks, and view the generated results.

The backend subsystem contains the core logic of the system, including the **Persona Creator, Task Creator, and Consensus Controller**, which coordinate persona generation, task execution, and consensus formation.

The API subsystem connects the platform with external services such as **LLM APIs for response generation and data scraping APIs** for collecting researcher information. The storage subsystem uses a PostgreSQL database to store personas, scraped research data, and other system information required for persona generation and session management.

3.3 Persistent Data Management

The system uses PostgreSQL as the primary database for persistent data storage. The database stores user accounts, generated personas, session information, and previously generated consensus results. This allows users to maintain their personalized configurations and revisit earlier analysis sessions.

In addition to system data, the platform stores academic metadata collected from Google Scholar using SerpAPI. This information includes researcher names, research topics, and publication details that are used to support the generation of researcher-based personas.

The collected researcher data is stored in JSON format within the database, allowing flexible storage and easy integration with the persona generation process [11].

3.4 Access Control and Security

The system requires users to **authenticate before accessing their personal sessions, saved personas, and generated results**. User authentication ensures that each user's data and preferences remain private and separate from other users. Authentication and access control mechanisms are essential components of secure web applications [12].

Some limited features may also be available to non-authenticated users; however, access to persistent data and personalized functionality requires a registered account.

4. Subsystem Services

The Consensia system relies on several external services to support **data collection and large language model interactions**. These services are integrated through the API layer and enable the system to generate responses, evaluate reasoning, and collect researcher information for persona creation.

4.1 Google Scholar API

The Google Scholar API is used to access publicly available academic information about researchers and their publications. This data is used to collect metadata such as author names, research topics, and publication details, which can later be used for creating researcher-based personas. Google Scholar provides access to a large collection of academic publications and researcher profiles that are commonly used for academic search and bibliographic analysis [13].

4.2 SerpAPI

SerpAPI is used to perform automated search queries and retrieve structured results from Google Scholar and other search engines. It simplifies the process of collecting academic data by returning results in a structured format such as **JSON**, which can then be stored and processed by the system for persona generation. SerpAPI provides a web-based API for accessing search engine results programmatically and returning them in structured formats suitable for software applications [14].

4.3 Google Gemini API

The Google Gemini API provides access to **large language models** used in the system. It is responsible for generating responses from different personas and assisting in the evaluation process performed by the judge model. The backend communicates with this API to send prompts and receive generated outputs during the multi-agent reasoning process. Google Gemini models are designed to support advanced language understanding, reasoning, and generative AI tasks through API-based integration [15].

5. Test Cases (functional and non-functional)

The following test cases are designed to validate the Consensia workspace, focusing on the asynchronous behavior of persona responses and the modular architecture of the system.

5.1 Functional Test Cases

TC 1.1: Persona Council Initialization

Procedure:
Select and add multiple personas (e.g., Lead Developer, SRE, and UI Designer) to the workspace.

Expected Result:
The horizontal Council Tray correctly displays the selected persona icons and metadata, preparing the system state for a multi-agent query.

TC 1.2: Asynchronous Persona Response Delivery

Procedure:
Submit a technical query and monitor the response cards displayed below the persona tray.

Expected Result:
Persona response cards populate independently as the respective LLM responses finish processing. Faster personas appear earlier without blocking the user interface.

TC 1.3: Context-Aware Reasoning

Procedure:
Analyze the outputs of two contrasting persona cards (e.g., a Security Specialist and a Performance Engineer).

Expected Result:
Each persona card produces reasoning aligned with its respective domain, demonstrating that the prompt-engineering layer correctly isolates perspectives.

TC 1.4: Judge Synthesis and Final Consensus

Procedure:
Observe the system behavior after all persona response cards reach the Completed state.

Expected Result:
The Judge LLM automatically processes the persona outputs and generates a final synthesized consensus block displayed at the bottom of the workspace.

TC 1.5: Session Restoration and State Durability

Procedure:
Close the browser session and log back into the system using the authenticated user account.

Expected Result:
The workspace is restored with the previously submitted question, selected personas, and all generated responses preserved through the PostgreSQL-backed session storage.

5.2 Non-Functional Test Cases

TC 2.1: UI Responsiveness under Inference Load

Procedure:
Interact with the persona tray or collapse response cards while multiple LLM requests are being processed.

Expected Result:
The React-based frontend remains responsive and interactive while asynchronous backend requests are in progress.

TC 2.2: Error Isolation (Fault Tolerance)

Procedure:
Simulate an API timeout or failure for a single persona response.

Expected Result:
The affected persona card displays an error or retry state while the remaining personas continue processing normally and the Judge LLM operates on available responses.

TC 2.3: Data Integrity and Access Control

Procedure:
Attempt to access a session identifier belonging to another user through the API.

Expected Result:
The system returns a 403 Forbidden response, confirming that session-level access control is properly enforced.

6. Consideration of Various Factors in Engineering Design

6.1 Constraints

Computational and Financial Constraints:
The system relies on external large language model services such as the **Google Gemini API**. Because these services operate under usage quotas and pricing models based on token consumption, the system must optimize prompt sizes and the number of persona interactions to remain within available usage limits [15].

Latency Constraints:
Multi-agent reasoning requires multiple LLM requests to be processed for a single task. This naturally increases response time. The system mitigates this through asynchronous processing and a non-blocking user interface, although total response time still depends on external API performance.

External Data Dependencies:
Persona creation may rely on academic data collected from **Google Scholar through services such as SerpAPI**. Therefore, the system depends on the availability, rate limits, and data formats of these external providers, which may change over time [13], [14].

6.2 Standards

RESTful API Design:
Communication between the frontend and backend follows RESTful API principles to ensure clear interfaces, maintainability, and scalability of the system architecture [16].

JSON Data Format (RFC 8259):
All data exchanged between system components, including LLM responses, persona data, and stored information, is formatted using JSON, which is a widely used standard for structured data exchange in web systems [11].

Data Privacy Standards (GDPR): As the system processes potentially sensitive professional data (CVs), it follows 'Privacy by Design' principles. This includes ensuring data isolation in the database and ensuring that external API calls to Gemini or SerpAPI do not persist personally identifiable information (PII) beyond the scope of the inference task.

Data Security Standards: All sensitive metadata within the PostgreSQL storage layer is handled using standard encryption protocols and Bcrypt hashing for credentials to prevent unauthorized data extraction from the persistent storage subsystem.

7. Teamwork Details

7.1 Contributing and functioning effectively on the team

The team divided the workload based on each member's strengths and technical skills to ensure efficient development. All members contributed to key components of the project, including system design, backend development, frontend implementation, and research tasks. Continuous communication was maintained throughout the development process to ensure that all team members remained aligned with project goals and progress.

7.2 Helping create a collaborative and inclusive environment

The team maintained regular communication through group messaging and scheduled discussions. In addition, weekly meetings were held with a doctoral researcher working with our supervisor, whose research is closely related to the project. These meetings provided technical guidance and helped the team coordinate progress and resolve challenges collaboratively.

7.3 Taking lead role and sharing leadership on the team

The team operates with a shared leadership approach rather than a single designated leader. Members take initiative when necessary, depending on the task or situation, allowing responsibilities to be distributed naturally while maintaining effective collaboration and decision-making.

8. Glossary

AI (Artificial Intelligence):
Systems designed to perform tasks that normally require human intelligence such as reasoning and decision-making.

LLM (Large Language Model):
A machine learning model trained on large text datasets to understand and generate natural language.

API (Application Programming Interface):
A mechanism that allows different software systems or services to communicate with each other.

PostgreSQL:
An open-source relational database management system used to store persistent system data.

REST API:
A web service architecture that allows communication between systems using standard HTTP methods.

JSON (JavaScript Object Notation):
A lightweight data format used for exchanging structured information between system components.

SerpAPI:
A service that provides structured access to search engine results and is used to retrieve academic data.

Google Gemini API:
An external API that provides access to Google's large language models used for response generation and evaluation.

CV (Curriculum Vitae):
A document describing a person's professional background and experience used for persona creation.

GDPR (General Data Protection Regulation): A legal framework that sets guidelines for the collection and processing of personal information from individuals.

Tangled Commit: A software version control contribution that contains multiple, logically independent changes (e.g., a bug fix, a refactoring task, and a new feature) within a single commit. These are difficult to review and label accurately, which serves as a primary challenge for the Consensia persona-matching research.

9. References

[1] T. Brown et al., “Language Models are Few-Shot Learners,” *NeurIPS*, 2020.

[2] Y. Du et al., “Improving Reasoning with Multi-Agent Debate,” *ICML*, 2024.

[3] L. Zheng et al., “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena,” *NeurIPS*, 2023.

[4] D. Norman, *The Design of Everyday Things*. Basic Books, 2013.

[5] S. Wu et al., “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation,” Microsoft Research, 2023.

[6] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th ed. Addison-Wesley, 2021.

[7] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2016.

[8] J. Wang et al., “CrewAI: Coordinating Role-Playing Autonomous Agents,” 2024.

[9] OpenAI, “OpenAI Evals Framework,” 2023.

[10] PostgreSQL Global Development Group, “PostgreSQL Documentation,” 2024.

[11] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, IETF, 2017.

[12] W. Stallings, *Effective Cybersecurity: A Guide to Using Best Practices and Standards*. Addison-Wesley, 2018.

[13] Google, “Google Scholar,” Available: https://scholar.google.com

[14] SerpAPI, “SerpAPI Documentation,” Available: https://serpapi.com

[15] Google DeepMind, “Gemini API Documentation,” Available: https://ai.google.dev

[16] R. Fielding and R. Taylor, “Architectural Styles and the Design of Network-Based Software Architectures,” Doctoral dissertation, University of California, Irvine, 2000.

[17] World Wide Web Consortium (W3C), “Web Content Accessibility Guidelines (WCAG) 2.1,” 2018.