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

1. Introduction

Modern software engineering decisions often require more than one perspective. Choices about architecture, implementation, testing, security, scalability, usability, and cost usually involve trade-offs between different priorities. In real teams, these decisions are made through discussions between people with different roles and expertise. However, such discussions can be time-consuming and difficult to reproduce in a structured way.

Consensia is a web-based intelligent discussion and decision-support system that uses large language models to simulate structured debates between expert personas. Instead of giving a single direct answer, the system allows users to define or generate multiple personas with different backgrounds and reasoning styles. Each persona responds to the user's question, participates in the discussion, and a judge model produces a final consensus based on the different arguments.

The main goal of Consensia is to make AI-assisted software engineering decisions more transparent and explainable [1]. A single AI response may overlook assumptions or ignore alternative solutions, while a multi-persona debate allows the user to compare different viewpoints before accepting a final recommendation. This makes the reasoning process easier to inspect and evaluate.

The final version of Consensia includes a debate workspace, persona management, file attachment support, persistent debate sessions, saved message history, favorite personas, user authentication, email verification, and password reset functionality. It also supports advanced persona generation from CV files and researcher-based evidence.

This report presents the final state of the Consensia project. It explains the system requirements, architecture, implementation details, testing results, maintenance plan, ethical considerations, team contributions, future improvements, and glossary.

2. Requirements Details

2.1 Functional Requirements

2.1.1 User Account and Authentication Requirements

● A user must be able to register by entering their name, email address, and password.

● A user must verify their email address before logging in with their account.

● A verified user must be able to log in and receive an access token for authenticated requests.

● A user must be able to reset their password through a verification code.

● A user must be able to view and update their profile information.

2.1.2 Persona Management Requirements

● A user must be able to create multiple AI personas for a debate.

● Each persona must have a name and description that define its background, expertise, priorities, and reasoning style.

● The system must provide default personas to help new users start quickly.

● A user must be able to manually add, edit, and remove personas before running a debate.

● An authenticated user must be able to save frequently used personas by adding them to favorites list.

● The system must support persona generation from uploaded CV files.

● The system must support researcher-based persona generation using researcher names.

● The system must limit the maximum number of personas in a debate to control response time and API cost.

2.1.3 Debate and Consensus Requirements

● A user must be able to enter a software engineering or technical question for discussion.

● The system must require at least one persona before starting a debate.

● Each selected persona must generate a response based on its own role, background, and reasoning style.

● The system must support multi-round debate so that personas can provide initial responses and follow-up reasoning.

● The system must generate a final judge consensus after collecting persona responses.

● The judge output must include a summary and reasoning explanation.

● The judge result must be treated as a recommendation, not as guaranteed truth.

● The system must support topic relevance and reasoning quality scores when available.

2.1.4 Attachment and Evidence Requirements

● A user must be able to attach supporting files to a debate.

● The system must support PDF and DOCX files by extracting their text and using it as debate evidence.

● The system must support image attachments for questions.

● The system must reject unsupported, unreadable, or oversized files with clear error messages.

● The system must store attachment metadata when a debate is saved in a session.

2.1.5 Debate Session and History Requirements

● An authenticated user must be able to create persistent debate sessions.

● A session must store the question, selected personas, debate result, timestamps, and message history.

● A user must be able to view, continue, and delete their own debate sessions.

● A user must not be able to access or modify another user's sessions.

● The system must store user messages, persona responses, judge outputs, round numbers, and attachment metadata.

● The system must maintain session continuity.

2.1.6 Export Requirements

● A user must be able to export debate results for later use.

● The export must include the question, personas, uploaded files, persona responses, judge summary, and judge reasoning.

● The system must support Markdown export.

● The system must support browser-based print or PDF export.

● The export should include topic relevance and reasoning quality scores when available.

● The export should include image previews when image attachments are available.

2.1.7 Admin and Usage Requirements

● The system must provide an admin interface for users with admin privileges.

● Admin users must be able to view statistics about users, sessions, messages, debate usage, and quota utilization.

● The system must enforce daily debate limits for guest and authenticated users.

● Admin users may have unrestricted usage for testing and monitoring purposes.

● Normal users must not be able to access admin endpoints.

2.1.8 Error Handling Requirements

● The system must validate user input before processing requests.

● The system must reject empty questions, missing personas, invalid login attempts, expired codes, unsupported files, and unauthorized session access.

● The system must return meaningful error messages instead of crashing.

● The system must handle LLM provider errors, scraping errors, and file extraction errors gracefully.

2.2 Non-Functional Requirements

2.2.1 Usability

● The user interface must be clear, modern, and easy to use.

● Users should be able to register, log in, create personas, run debates, view results, manage sessions, and export outputs without technical knowledge.

● The debate page must clearly separate the user question, persona responses, and judge consensus.

● The system must show loading states during long operations such as debate generation, CV extraction, and researcher persona creation.

2.2.2 Reliability

● The system must handle backend, database, file-processing, and LLM errors without crashing.

● The system must maintain consistent data for users, sessions, messages, attachments, favorites, and quotas.

● Users must only be able to access their own account data and debate sessions.

2.2.3 Performance

● The system should generate persona responses and judge consensus efficiently enough for interactive use.

● The backend should use asynchronous processing where possible for LLM calls.

● The system must limit the number of personas and prompt sizes to reduce latency and API cost.

● The system should not promise a fixed response time because performance depends on the LLM provider, file size, number of personas, and network conditions.

2.2.4 Security and Privacy

● The system must store API keys, JWT secrets, SMTP credentials, database URLs, and provider settings in environment variables.

● User passwords must be hashed before being stored.

● Protected endpoints must require authentication.

● Admin endpoints must require admin privileges.

2.2.5 Maintainability

● The system must follow a modular full-stack architecture.

● The frontend, backend, database, authentication, session management, LLM orchestration, persona generation, export, and admin features should be separated clearly.

● The backend should use structured schemas for request and response validation.

● The frontend should use reusable React components.

● LLM provider logic should be separated so that different providers can be added or changed easily.

2.2.6 Scalability and Configurability

● The system must support configurable LLM providers such as OpenAI.

● Model names, API keys, persona limits, debate limits, prompt limits, and context sizes must be configurable through environment variables.

● The system should allow future support for new persona sources, evidence types, and LLM providers.

2.2.7 Portability

● The system must be runnable using Docker Compose.

● Docker Compose must run the frontend, backend, and PostgreSQL database together.

● This requirement helps team members run the project consistently on different operating systems such as Windows and macOS.

2.2.8 Cost Control

● The system must limit unnecessary LLM usage.

● The system must control cost through daily debate limits, maximum persona limits, maximum prompt sizes, and bounded session context.

● The system should support simulated LLM responses during development when real API keys are not configured.

2.3 Pseudo Requirements

2.3.1 Version Control

● GitHub must be used as the main version control platform.

● Development may happen on feature branches before being merged into the main branch.

2.3.2 Technology Stack

● The frontend must be implemented using React, TypeScript, Vite, and Tailwind CSS.

● The backend must be implemented using Python and FastAPI.

● PostgreSQL must be used as the main database.

● SQLAlchemy and Alembic must be used for database modeling and migrations.

● Docker Compose must be used for local development.

● The system must support OpenAI as an LLM provider.

● The system may use SerpAPI for researcher-based persona generation.

2.3.3 Configuration

● The backend must read configuration values from environment variables and .env files.

● Real API keys and secrets must not be committed to GitHub.

● The repository should provide an environment template for developers.

● The frontend must support same-origin API calls through the Vite proxy during development.

2.3.4 Testing and Validation

● The system must be tested for registration, email verification, login, password reset, persona creation, debate execution, attachment upload, session management, export, and admin access.

● The system must be tested with invalid cases such as duplicate registration, invalid login, unverified login, missing personas, empty questions, unsupported files, quota exhaustion, and unauthorized access.

● The system must be tested in Docker to confirm that the frontend, backend, and database work together correctly.

● The system must be tested with both real LLM providers and simulated responses.

3. Final Architecture and Design Details

3.1 Overview

Consensia follows a layered client-server architecture [2]. At the highest level, the system consists of the frontend layer, backend layer, persistence layer, AI/LLM processing layer, and external services layer. Each layer has a separate responsibility, which helps keep the system organized and easier to maintain.

The frontend layer is implemented using React, Vite, TypeScript, and Tailwind CSS. It provides the user-facing interface where users can register, log in, manage personas, enter debate questions, upload supporting files, view generated debate results, access previous sessions, and export outputs. The frontend communicates with the backend through REST API calls.

The backend layer is implemented using FastAPI. It acts as the central coordinator of the system by handling API routing, request validation, authentication, user management, session management, attachment processing, researcher persona generation, quota control, and debate orchestration. The

backend is also responsible for communicating with the persistence layer, AI/LLM processing layer, and external services.

The persistence layer uses PostgreSQL to store long-term application data such as users, authentication records, debate sessions, debate messages, uploaded file metadata, favorite personas, and rate-limit records. This allows users to save debates, revisit previous sessions, and reuse persona configurations.

The AI/LLM processing layer is responsible for the intelligent reasoning part of Consensia. It generates persona-specific responses, supports multi-round debate, evaluates persona contributions, and produces the final judge consensus. External services such as OpenAI, SerpAPI, and Google Scholar are used when configured for LLM inference and researcher-based persona generation.

Overall, this architecture supports the main goal of Consensia: generating explainable multi-persona discussions and final consensus outputs instead of producing a single direct chatbot response.

3.2 Subsystem Decomposition

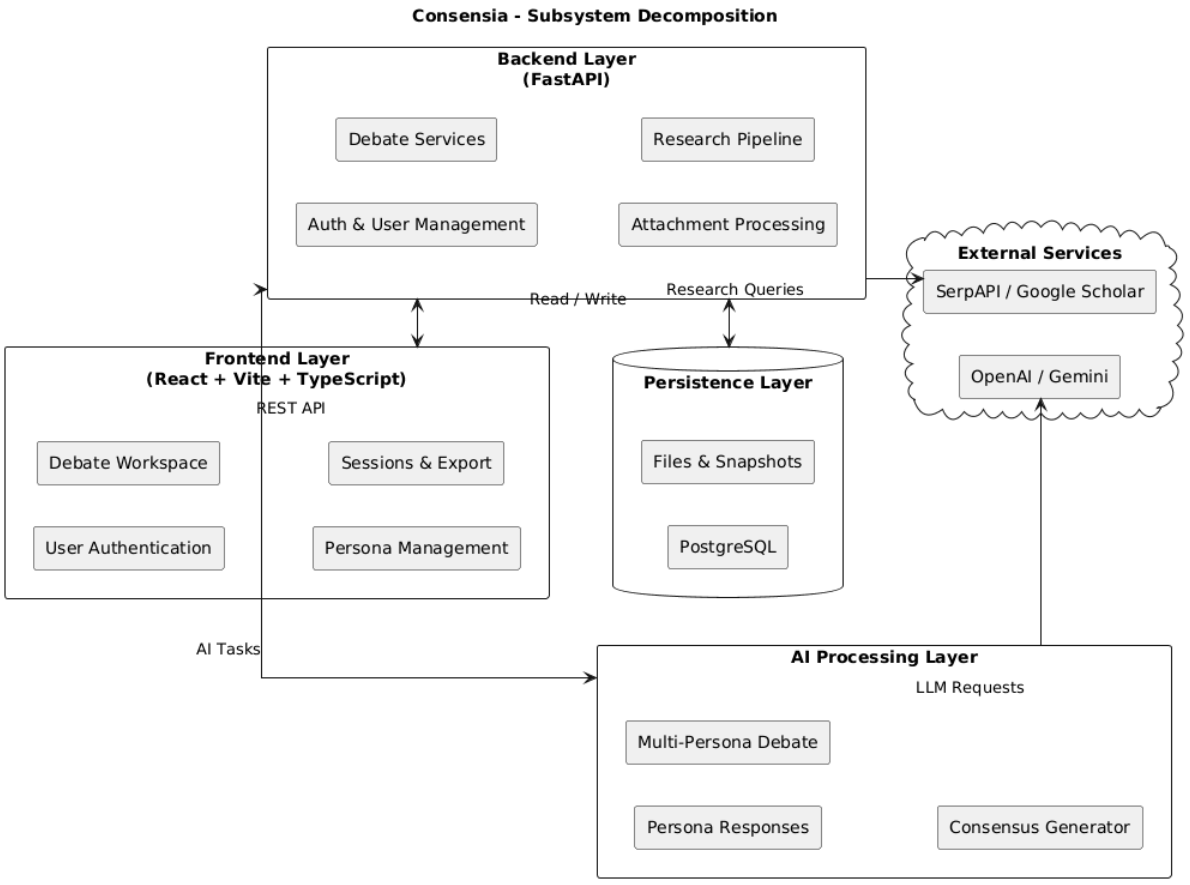

Figure 3.1: High-Level Subsystem Decomposition Diagram

The high-level subsystem decomposition diagram shows the main layers of Consensia and how they interact with each other. It includes the frontend layer, backend layer, persistence layer, AI processing layer, and external services. The diagram shows that the user interacts with the frontend, while the frontend communicates with the backend through REST API calls. The backend then coordinates database operations, AI task execution, and external service communication.

The frontend layer contains the main user-facing modules, including the debate workspace, user authentication, persona management, and session/export functionality. These modules allow the user to ask questions, manage personas, run debates, and review saved results.

The backend layer contains the main application services, including debate services, authentication and user management, attachment processing, and the research pipeline. This layer acts as the central controller of the system. It receives requests from the frontend, validates them, processes user data, communicates with the database, and sends AI-related tasks to the AI processing layer.

The persistence layer represents the PostgreSQL database and stored application data. It stores debate sessions, user data, uploaded file metadata, saved research snapshots, favorite personas, and other records needed for session continuity and traceability [3].

The AI processing layer is responsible for multi-persona debate generation, persona responses, and consensus generation. It communicates with external LLM services such as Google Gemini and OpenAI (including Azure-hosted OpenAI-compatible deployments) through backend-controlled requests.

The external services layer includes Google Gemini, OpenAI, SerpAPI, and Google Scholar-related services. These services are outside the Consensia system boundary but are used for LLM response generation and researcher-based persona creation.

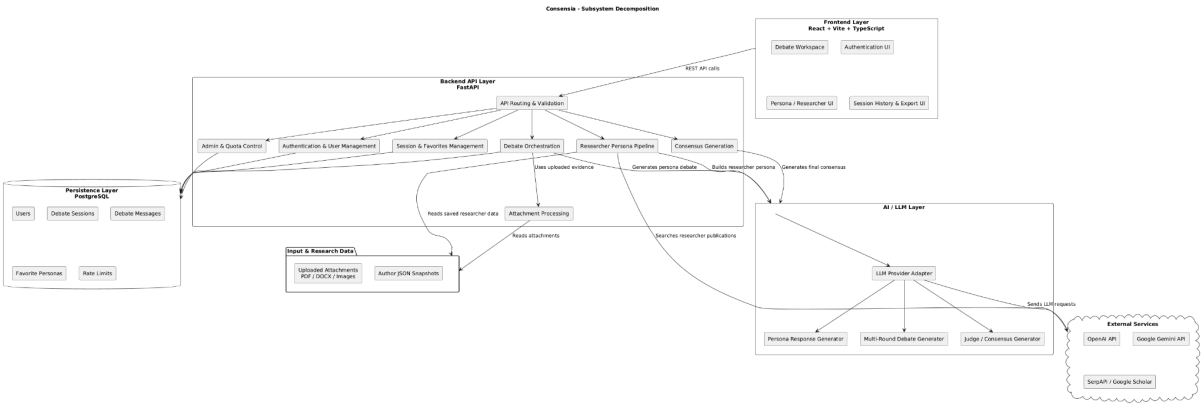

Figure 3.2: Detailed Subsystem Decomposition Diagram

The detailed subsystem decomposition diagram presents the internal organization of Consensia in a more focused way. It shows the frontend modules for debate interaction, authentication, persona management, and session/export features; the backend modules for request validation, user management, session handling, debate orchestration, attachment processing, researcher persona generation, consensus generation, and admin/quota control; the persistence layer for PostgreSQL-based storage of users, sessions, messages, favorite personas, and rate-limit data; the input and research data area for

uploaded files and researcher snapshots; and the AI/LLM layer for provider adaptation, persona response generation, multi-round debate generation, and judge consensus generation. Overall, the diagram emphasizes that all database access, LLM communication, file processing, and external service interaction are routed through the backend, which improves both security and maintainability.

3.2.1 Frontend Layer

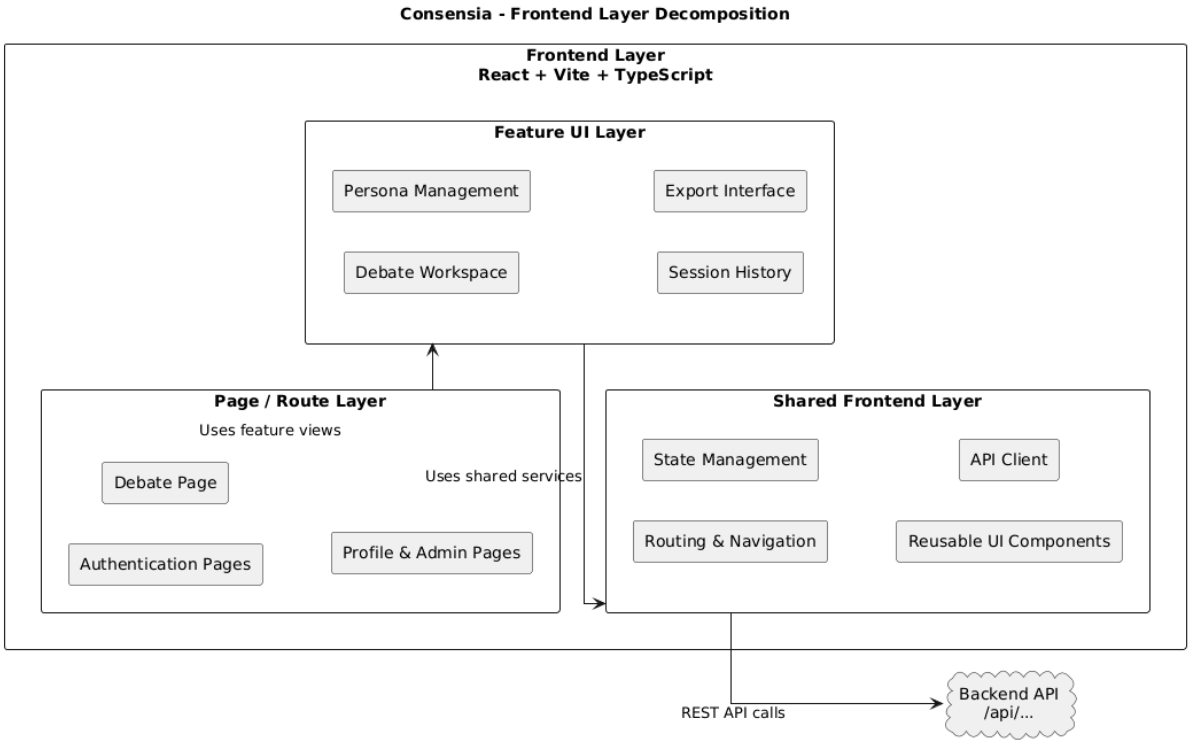

Figure 3.3: Basic Frontend Layer Diagram

● **Purpose:**
The frontend layer is the user-facing interface of Consensia. It allows users to register and log in, manage personas, enter debate questions, upload supporting files, run debates, view generated results, access previous sessions, and export debate outputs.

● **Technology:**
The frontend is implemented using **React, Vite, TypeScript, and Tailwind CSS**.

● **Modules:**
a. **Page / Route Layer:** Contains the main application pages, including the

debate page, authentication pages, profile page, and admin pages. This layer organizes the navigation flow of the application.

b. **Feature UI Layer:** Contains the main user-facing features, including the debate workspace, persona management, session history, and export interface. These modules allow users to create and manage debates, reuse personas, review previous sessions, and export results.

c. **Shared Frontend Layer:** Contains reusable frontend services and components, including state management, routing and navigation, the API client, and reusable UI components. These shared modules support the feature and page layers.

d. **API Client:** Handles communication between the frontend and backend by sending REST API requests to the FastAPI backend.

● **Data Flow:**

a. **Page / Route Layer → Feature UI Layer:** Pages use feature views such as the debate workspace, persona management, session history, and export interface.

b. **Feature UI Layer → Shared Frontend Layer:** Feature components use shared services such as state management, reusable UI components, routing, and the API client.

c. **Frontend → Backend:** The frontend sends REST API calls for authentication, debate generation, persona management, file upload, session retrieval, profile actions, admin data, and export-related data.

d. **Backend → Frontend:** The backend returns authentication responses, saved sessions, debate results, persona outputs, judge consensus, profile data, admin statistics, error messages, and status updates.

3.2.2 Backend Layer

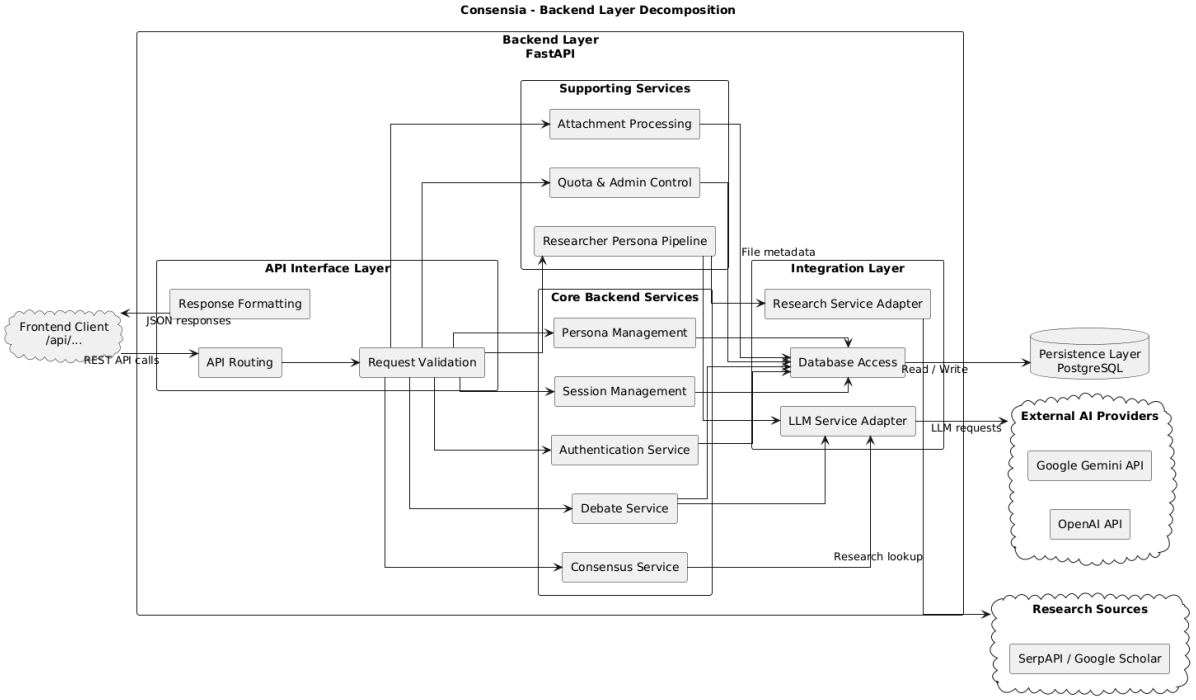

Figure 3.4: Backend Layer Diagram

● **Purpose:**
The backend layer is the central control layer of Consensia. It receives requests from the frontend, validates input, manages authentication, coordinates debate generation, processes attachments, communicates with AI providers and research services, and stores or retrieves data from the database.

● **Technology:**
The backend is implemented using **Python and FastAPI**.

● **Modules:**
a. **API Interface Layer:** Handles REST API communication with the frontend. It includes API routing, request validation, and response formatting.

b. **Core Backend Services:** Contains the main application logic, including authentication service, session management, persona management, debate service, and consensus service.

c. **Supporting Services:** Provides additional backend functionality such as attachment processing, quota and admin control, and researcher persona pipeline.

d. **Integration Layer:** Connects the backend to external and internal services. It includes the data access module for PostgreSQL, the LLM service adapter that routes requests to the configured provider (OpenAI-compatible or Google Gemini), and the research service adapter for SerpAPI or Google Scholar-based lookup.

● **Data Flow:**

a. **Frontend → Backend:** The frontend sends REST API calls for authentication, debate execution, persona management, session operations, file uploads, admin data, and researcher-persona generation.

b. **API Interface Layer → Core Backend Services:** Incoming requests are routed and validated before being passed to the appropriate backend service.

c. **Core Backend Services → Supporting Services:** Debate and persona-related requests may use attachment processing, quota checks, admin control, or researcher persona generation.

d. **Backend → Persistence Layer:** The backend reads and writes users, authentication records, debate sessions, messages, attachments, favorite personas, and quota records in PostgreSQL.

e. **Backend → External AI Providers**: The LLM service adapter sends model requests to the configured LLM provider for persona responses, debate generation, and consensus generation.

f. **Backend → Research Sources:** The research service adapter sends lookup requests to SerpAPI or Google Scholar-related sources for researcher-based persona generation.

g. **Backend → Frontend:** The backend returns JSON responses containing authentication results, generated debate outputs, judge consensus, saved sessions, profile data, admin statistics, errors, and status messages.

3.2.3 Persistence / Database Layer

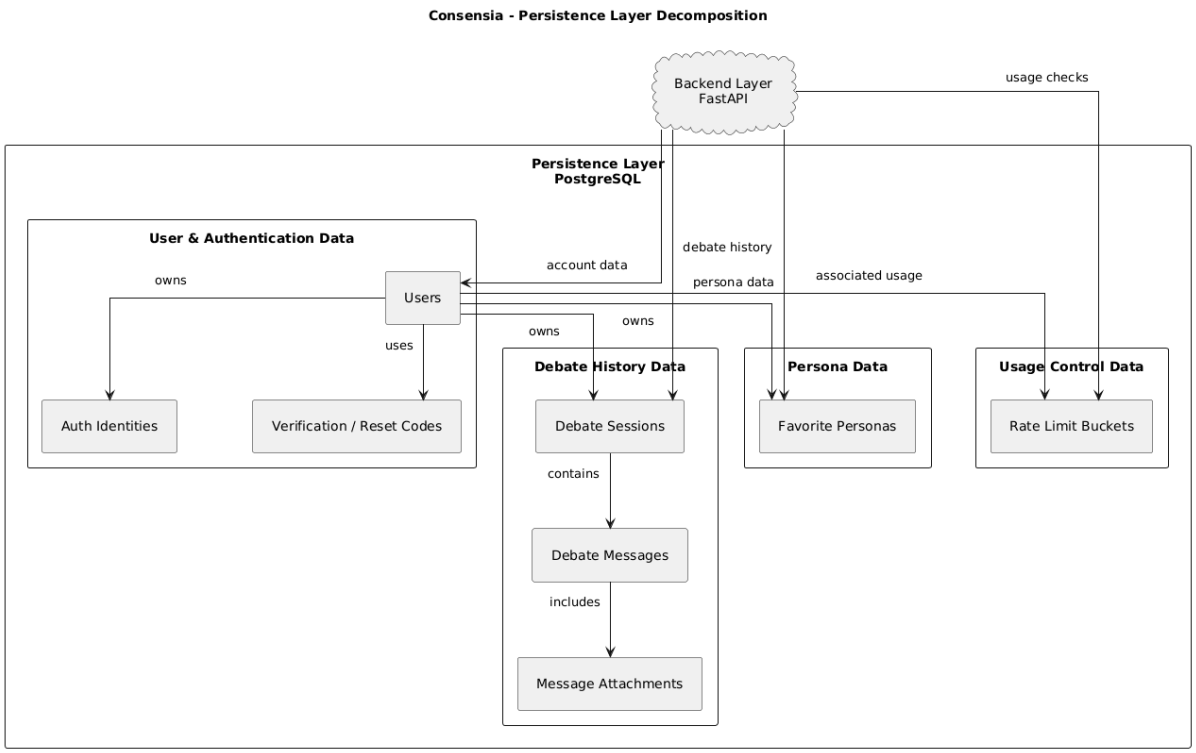

Figure 3.5: Database Layer Diagram

● **Purpose:**

The persistence layer stores the long-term data of Consensia. It allows the system to keep user accounts, authentication records, debate history, message attachments, favorite personas, and usage-limit information across different sessions.

● **Technology:**

The persistence layer uses PostgreSQL as the main database [3]. The database models are defined using an ORM with SQLAlchemy, which allows backend entities to be represented as Python classes [4]. Alembic is used for database migrations, so changes to the database schema can be versioned and applied consistently during development [3].

● **Components:**

a. **User & Authentication Data:** Stores user accounts, authentication identities, email verification codes, and password reset codes. This data supports registration, login, email verification, password recovery, and account management.

b. **Debate History Data:** Stores debate sessions, debate messages, and message attachments. Debate sessions contain the main debate information, while debate messages store user questions, persona responses, judge outputs, round data, and related metadata.

c. **Persona Data:** Stores favorite personas saved by authenticated users. This allows users to reuse frequently used persona profiles in later debates.

d. **Usage Control Data:** Stores rate-limit buckets used to track daily debate usage for guest and authenticated users. This helps control LLM usage and prevent excessive API calls.

● **Data Flow:**

a. **Backend → PostgreSQL:** The backend writes account data, session records, debate messages, uploaded attachment metadata, favorite personas, and rate-limit information to the database through SQLAlchemy ORM models.

b. **PostgreSQL → Backend:** The database returns user records, authentication status, saved sessions, debate history, favorite personas, and usage information when requested by the backend.

c. **Users → Debate Data:** A user can own multiple debate sessions, and each debate session can contain multiple debate messages and message attachments.

d. **Users → Persona Data:** A user can save multiple favorite personas for reuse in future debate sessions.

e. **Users → Usage Data:** A user or guest identity is associated with rate-limit records that track allowed debate usage.

● Design Note:

The frontend does not directly access the database. All database operations are handled through the backend layer using SQLAlchemy ORM models, while Alembic manages schema migrations. This protects stored data, keeps database logic centralized, and ensures that authentication and ownership checks are applied before users can access saved sessions, messages, or profile information [4].

3.2.4 AI / LLM Processing Layer

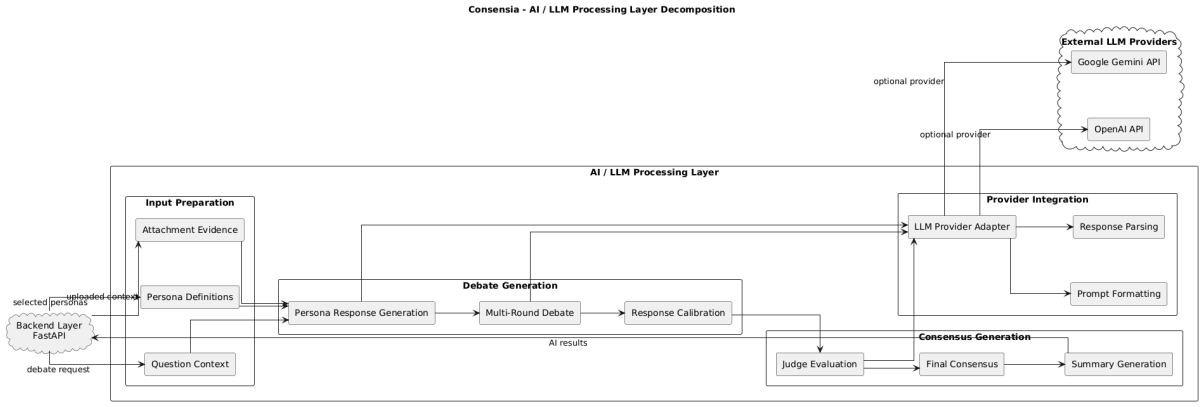

Figure 3.6: LLM Processing Layer Diagram

● **Purpose:**
The AI/LLM processing layer is responsible for the intelligent reasoning tasks of Consensia. It receives prepared debate requests from the backend, generates persona-specific responses, supports multi-round debate, evaluates response quality, and produces the final judge consensus. The system is designed to be provider-agnostic, integrating with Azure OpenAI and Google Gemini depending on the environment configuration. Each persona receives a system prompt embedding their background description, while the judge model utilizes a lower temperature setting to ensure consistency across synthesized outputs.

● **Design Note:**
Although the AI/LLM processing layer is coordinated by the backend, it is shown as a separate decomposition diagram because including all AI modules inside the backend diagram would make the architecture visually crowded and ambiguous. Separating this layer makes the debate-generation pipeline, provider integration, and consensus-generation flow easier to understand.

● **Modules:**
a. **Input Preparation:** Prepares the information needed for AI processing. This includes the user's question context, selected persona definitions, and uploaded attachment evidence.

b. **Debate Generation:** Generates the main debate outputs. It includes persona response generation, multi-round debate handling, and response calibration.

c. **Provider Integration:** Connects the system to external LLM providers through an LLM provider adapter. It also handles prompt formatting and response parsing so that provider-specific differences are hidden from the rest of the system.

d. **Consensus Generation:** Produces the final judge output. It includes judge evaluation, final consensus generation, and summary generation.

e. **External LLM Providers:** Azure OpenAI (OpenAI-compatible), direct OpenAI, and Google Gemini are optional external providers for model inference when configured.

● **Data Flow:**

a. **Backend → Input Preparation:** The backend sends the debate request, selected personas, uploaded context, and question context to the AI/LLM processing layer.

b. **Input Preparation → Debate Generation:** Persona definitions, attachment evidence, and question context are used to generate persona responses and multi-round debate outputs.

c. **Debate Generation → Provider Integration:** Debate-generation modules send formatted prompts through the LLM provider adapter when external model calls are needed.

d. **Provider Integration → External LLM Providers:** The provider adapter sends LLM requests to the configured provider endpoint.

e. **External LLM Providers → Provider Integration:** The selected provider returns generated model responses, which are parsed and normalized by the provider integration module.

f. **Debate Generation → Consensus Generation:** Persona responses, debate rounds, and calibration results are passed to the judge evaluation and consensus generation modules [5].

g. **Consensus Generation → Backend:** The final consensus, summary, reasoning, and related AI results are returned to the backend so they can be stored in the database and displayed in the frontend.

3.3 Layer-to-Layer Communication

This subsection summarizes how the layers communicate with each other.

● **Frontend → Backend:**
The frontend sends REST API requests for authentication, debate execution, persona management, file upload, session history, profile actions, export, and admin functions.

● **Backend → Persistence Layer:**
The backend reads and writes user data, authentication records, debate sessions, debate messages, attachments, favorite personas, and rate-limit data through SQLAlchemy ORM and PostgreSQL [3, 4].

● **Backend → AI / LLM Processing Layer:**
The backend sends prepared debate tasks, selected personas, question context, uploaded evidence, and session context to the AI/LLM processing layer.

● **AI / LLM Processing Layer → External LLM Providers:**
The LLM provider adapter sends requests to the configured provider and receives generated model responses.

● **Backend → External Research Services:**
For researcher-based persona generation, the backend may send lookup requests to SerpAPI or Google Scholar-related services.

● **Backend → Frontend:**
The backend returns authentication results, generated persona responses, judge consensus, saved session data, profile data, admin statistics, errors, and status updates.

3.4 Advantages of the Architecture

● **Modularity:**
Each layer has a clear responsibility, which makes the system easier to understand and maintain.

● **Maintainability:**
Frontend, backend, database, AI processing, and external integrations are separated, so changes in one part do not require rewriting the whole system.

● **Security:**
The frontend does not directly access the database, LLM providers, API keys, or external services. Sensitive operations are handled by the backend.

● **Extensibility:**
New LLM providers, persona-generation methods, attachment types, scoring methods, or export formats can be added in the future.

● **Persistence and Traceability:**
Debate sessions, messages, personas, attachments, and judge outputs are stored in PostgreSQL, allowing users to revisit and review previous debates.

● **Explainability:**
The system separates persona responses, debate rounds, calibration scores, and judge consensus, making the final recommendation easier to inspect.

● **Portability:**
The system can be run locally using Docker Compose, which helps team members work consistently across different operating systems.

---

4. Development / Implementation Details

4.1 Technology Stack Overview

Consensia is a full-stack web application consisting of a React/TypeScript frontend, a FastAPI Python backend, and a PostgreSQL database, all containerized and orchestrated via Docker Compose.The backend integrates asynchronously with external LLM providers (OpenAI-compatible APIs, including Azure OpenAI, and Google Gemini) through a unified service class that resolves which provider to use at startup based on environment configuration. SQLAlchemy handles the ORM layer and Alembic manages database migrations.

4.2 Backend Implementation

4.2.1 Application Startup and Routing

On startup, the application automatically runs database migrations. If migrations fail, the application refuses to start entirely. Four sub-routers are mounted: authentication, debate sessions, persona favorites, and admin.

4.2.2 Core Debate Endpoints

The two central API endpoints are POST /api/consensus and POST /api/debate. The key implementation decision is concurrency: all persona LLM calls for a given request are dispatched simultaneously, so total response time is bounded by the slowest single call rather than growing linearly with the number of personas. The judge consensus is generated sequentially after all persona answers are collected. [6]

The /api/debate endpoint extends this to support multi-round structured debates, where each subsequent round's persona prompts are augmented with the previous round's full transcript. Uploaded documents have their text extracted server-side and injected into the prompt as evidence.

4.2.3 LLM Service and Persona Orchestration

The LLMService class is the core of the application. If no API keys are present, it falls back to a simulated mode that returns placeholder responses, allowing frontend development without incurring API costs. Each persona receives a system prompt embedding their name, description, and optionally their source material such as a researcher profile.

For multi-persona debates, a two-pass QA calibration runs automatically. Before answers are generated, each persona is scored on how relevant the question is to their domain. After answers are generated, each persona's arguments are scored for reasoning quality. These scores are combined into normalized per-persona weights passed to the judge, biasing its synthesis toward the most relevant and highest-quality contributors [7].

The judge is not fed only raw persona text: topic-relevance and reasoning-quality scores are turned into normalized per-persona weights so the synthesis emphasizes stronger, more on-topic contributions. Session follow-ups

reuse prior context through a rolling window of the most recent text (truncated to a fixed word/token budget) instead of attaching the entire transcript every time, which keeps prompts bounded while preserving continuity across turns.

4.2.4 Authentication

The system supports both local email/password login and Google OAuth. Local accounts hash passwords with pwdlib's recommended hasher (Argon2id) and require email verification via a time-limited code sent over SMTP [8]. Both flows return a signed JWT access token used to authenticate subsequent requests. Authenticated users get full session persistence; their debate history, personas, and results are stored and retrievable across sessions.

4.2.5 Data Models

The key database models are User, AuthIdentity (supports multiple auth providers per account), DebateSession (persists the question, personas, and result), DebateMessage (individual chat entries with role and round metadata), DebateRateBucket (per-IP and per-user daily quota tracking), and FavoritePersona (saved persona presets).

4.3 Frontend Implementation

The frontend is organized into pages and shared components. The Debate page is the most complex view, managing the active persona list, chat thread, file attachments, and session sidebar together. The Persona Panel lets users add, edit, and remove personas with default presets pre-loaded. After a debate is submitted, responses are appended to the chat thread with distinct roles (persona, judge, user, and system) giving the UI a structured foundation for rendering each message type differently. All API calls go through a centralized wrapper that automatically attaches the authentication token to every request.

4.4 Deployment

Docker Compose runs all three services locally with hot-reload enabled, making it practical as a development environment without any rebuilds. The frontend

proxy forwards all API requests to the backend internally so the browser communicates with a single origin.

For production, the frontend is deployed as a static build to Netlify or GitHub Pages and the backend is hosted on a platform such as Render. An environment variable at build time points the frontend at the production backend URL, and CORS is configured on the backend to allow only the listed frontend origins.

4.5 Configuration

All runtime configuration is managed through environment variables loaded from a .env file. The critical values are the database URL, JWT secret key, LLM provider and API keys, CORS origins, SMTP credentials, and the Google OAuth client ID. An env.template file documents all required variables with placeholder values, and the actual .env is never committed to the repository.

4.6 The Research Component: Classifying Tangled Commits

While the primary focus of the project is the Consensia web platform, a dedicated research study was conducted using the system's infrastructure to evaluate how effectively language models can perform complex code annotation tasks. The study replicated the manual annotation task from the paper *"A fine-grained data set and analysis of tangling in bug reports"* [9].

In software engineering, a "tangled commit" occurs when a developer submits multiple, unrelated changes within the same commit. The task requires reading raw code diffs and classifying the purpose of each individual line. Google Gemini 2.5 Flash was utilized specifically for this research phase, as its large context window is optimized for parsing thousands of lines of code while adhering to strict JSON formatting rules.

Identified AI Limitations: Language models process text sequentially and inherently struggle to interpret the two-dimensional visual layout of code blocks. Early iterative testing revealed two major spatial reading errors. First, the models would erroneously group unrelated empty lines and global imports into functional categories simply because they were physically close to actual code changes on the screen. Second, when the models were instructed to strictly

ignore empty lines to solve the first issue, they began fracturing cohesive functions by leaving out closing brackets and inner spaces, which severely degraded accuracy. [9]

The Dual-Prompt Architecture:

To resolve these limitations, the system's prompt instructions were divided into two distinct components to separate subjective decision-making from objective spatial logic:

1. **The Persona Prompt:** This injects the nuanced JSON profile generated by the scraping pipeline. By providing the model with the researcher's specific domain_focus_weights, biases, and annotation_rules, this prompt controls *how* the AI subjectively judges the code.
2. **The Task Prompt:** This strictly handles the spatial mapping and structural logic to prevent the AI's reading errors. It mechanically forces the model to follow these sequential steps:
    - It evaluates the file path first; if terms like "test" or "mock" are present, the code is assumed to be testing logic, preventing it from being misclassified as a bug fix.
    - It isolates comments, imports, and empty spaces into their own respective categories before the core logic is even analyzed.
    - It forces the model to write out a step-by-step thought process explaining exactly where cohesive code blocks start and end before outputting any final line numbers.
    - It strictly dictates that cohesive functions must not be broken apart, ensuring inner blank lines or closing brackets are treated as part of the surrounding function.

4.6.1 Research Results and the Impact of Annotator Styles

The dual-prompt architecture was formally tested using a 5-Fold Cross-Validation setup. To prevent the model from becoming confused by conflicting historical labels, a sampling script randomly selected exactly two highly varied code examples from the training data for the model to use as clean calibration examples in each fold.

Crucially, the cross-validation tests revealed that the code inputs behaved fundamentally differently depending on the original human annotator's personal labeling choices. Each annotator possessed a distinct subjective style, and the experiment evaluated whether injecting the weighted JSON persona could force the AI to accurately match those varying human styles. The tests were executed under two conditions: a Control setup using a generic expert instruction, and a Persona setup utilizing the scraped researcher profiles.

- **Matching a Structured Style (Dataset 1 - A.T.):** This annotator possessed a highly structured and consistent subjective style [9]. The Persona setup achieved a Cohen's Kappa ($\varkappa$) of 0.6304 and an F2-Score of 0.7991, almost matching the Control setup's Kappa of 0.6833.

- **Matching an Ambiguous Style (Dataset 2 - S.H.):** This annotator's inputs were highly ambiguous, messy, and possessed a much greater tangling density [9]. In this scenario, the Control setup ($\varkappa$: 0.5870) slightly outperformed the Persona setup ($\varkappa$: 0.5451). The results indicate that when dealing with highly complex and erratic human labeling choices, a generalized expert baseline adapts more flexibly to the code than a strictly constrained psychological persona.

For both of these annotators the persona method was outperformed by the control method. Since LLMs are already trained, the personafication method did not help with the annotation task. Overall, the evaluation confirmed that Personas are still able to match distinct human annotation styles on well-structured datasets.

---

5. Test Cases and Results

This section presents 65 test cases covering the functional and non-functional requirements of Consensia. All tests are written at the integration level and can be executed by a test engineer without access to the source code. Each test case describes what is being verified, the steps to follow, and the expected outcome.

All tests were executed against the full Docker Compose deployment with Azure OpenAI configured as the primary LLM provider.

5.1 Severity Levels

The following table defines the three severity levels used to classify test cases in this document. [10, 11, 12]

Severity	Definition
Critical	The functionality under test is central to the system. A failure at this level means a core feature does not work, data may be lost or exposed, or the application cannot be used for its primary purpose. These issues must be resolved before the system can be considered functional.
Major	An important feature or quality requirement does not behave as expected, but the system remains partially usable. These issues have a significant impact on user experience or system reliability and should be addressed in the same release.
Minor	A low-impact issue that does not prevent the system from being used. The affected functionality is secondary or has an acceptable workaround. These issues can be addressed in a future maintenance cycle.

5.2 Functional Test Cases

The following 50 test cases verify that the system's features behave correctly under normal and boundary conditions.

TC-AUTH-01 – User Registration and Email Verification

Test ID	TC-AUTH-01	Category	Functional	Severity	Critical
Objective	This test case verifies that a new user can register with a valid email and password, receive a verification code, and complete the verification process to activate their account.				
Steps	1. Navigate to the registration page and fill in a valid email address, full name, and password. 2. Submit the registration form. 3. Open the email inbox associated with the registered address and locate the verification code. 4. Navigate to the email verification page and enter the received code. 5. Confirm the verification and observe the result.				
Expected	The account is created and, after entering the correct verification code, the user is marked as verified and redirected to the main application.				
Date – Result	May 2026 — **PASS**				

TC-AUTH-02 – Login with Valid Credentials

Test ID	TC-AUTH-02	Category	Functional	Severity	Critical
Objective	This test case verifies that a registered and verified user can log in with their correct email and password and gain access to the application.				
Steps	1. Navigate to the login page. 2. Enter a registered and verified email address with the correct password. 3. Submit the login form and observe the result.				
Expected	The user is authenticated and redirected to the debate workspace.				
Date – Result	May 2026 — **PASS**				

TC-AUTH-03 – Login with an Incorrect Password is Rejected

Test ID	TC-AUTH-03	Category	Functional	Severity	Major
Objective	This test case verifies that the system correctly rejects a login attempt when the password entered does not match the account.				
Steps	1. Navigate to the login page. 2. Enter a registered email address with an incorrect password. 3. Submit the form.				
Expected	Login fails and an error message is displayed. The user is not authenticated and remains on the login page.				
Date – Result	May 2026 — **PASS**				

TC-AUTH-04 – Unverified Account Cannot Log In

Test ID	TC-AUTH-04	Category	Functional	Severity	Major
Objective	This test case verifies that a user who has registered but not completed email verification cannot access the application.				
Steps	1. Register a new account but do not complete the email verification step. 2. Attempt to log in with the credentials of the unverified account.				
Expected	Login is blocked. The system informs the user that their email address has not been verified.				
Date – Result	May 2026 — **PASS**				

TC-AUTH-05 – Password Reset via Email Code

Test ID	TC-AUTH-05	Category	Functional	Severity	Major
Objective	This test case verifies that a registered user can reset a forgotten password by requesting a code via email and setting a new password.				
Steps	1. Submit a password reset request using a registered email address. 2. Retrieve the reset code from the email inbox. 3. Enter the code along with a new password on the reset page. 4. Attempt to log in using the new password.				
Expected	The password is updated successfully. The old password no longer grants access and the new password works correctly.				
Date – Result	May 2026 — **PASS**				

TC-AUTH-06 – Expired Session Requires Re-authentication

Test ID	TC-AUTH-06	Category	Functional	Severity	Minor
Objective	This test case verifies that an expired authentication session is not accepted and the user must log in again to continue.				
Steps	1. Log in and allow the session to expire without any activity. 2. Attempt to access a page that requires authentication.				
Expected	The session is rejected and the user is required to log in again.				
Date – Result	May 2026 — **PASS**				

TC-PER-01 – Create a Persona Manually

Test ID	TC-PER-01	Category	Functional	Severity	Critical
Objective	This test case verifies that a user can create a persona by manually providing a name and a background description.				
Steps	1. Open the Persona Panel and select the manual creation option. 2. Enter a persona name and a background description. 3. Save the persona. 4. Verify that the persona appears in the persona grid.				
Expected	The persona is saved and displayed in the grid with the correct name and description.				
Date – Result	May 2026 — **PASS**				

TC-PER-02 – Create a Persona from a PDF CV

Test ID	TC-PER-02	Category	Functional	Severity	Critical
Objective	This test case verifies that a user can upload a PDF CV and the system generates a persona based on the content of the document.				
Steps	1. Select the CV upload option in the persona creation flow. 2. Upload a valid PDF CV file. 3. Wait for the system to process the file. 4. Review the generated persona name, title, and description.				
Expected	The system extracts relevant information from the CV and generates a persona with a name, title, and description. The result is displayed for the user to confirm.				
Date – Result	May 2026 — **PASS**				

TC-PER-03 – Create a Persona from a DOCX CV

Test ID	TC-PER-03	Category	Functional	Severity	Critical
Objective	This test case verifies that a user can upload a DOCX CV file and the system generates a persona based on its content.				
Steps	1. Select the CV upload option and upload a valid DOCX file. 2. Wait for the system to process the file. 3. Review the generated persona fields.				
Expected	The system reads the document and generates a valid persona with a name, title, and description.				
Date – Result	May 2026 — **PASS**				

TC-PER-04 – Create a Persona from a Researcher's Profile

Test ID	TC-PER-04	Category	Functional	Severity	Critical
Objective	This test case verifies that a user can search for a researcher by name and the system generates a persona grounded in their published work.				
Steps	1. Select the researcher search option and enter a researcher's name. 2. Submit the search and wait for the pipeline to complete. 3. Review the generated persona card.				
Expected	The system retrieves the researcher's profile and publication data, processes it, and generates a persona description based on their actual research background.				
Date – Result	May 2026 — **PASS**				

TC-PER-05 – Unsupported File Type is Rejected for CV Upload

Test ID	TC-PER-06	Category	Functional	Severity	Major
Objective	This test case verifies that the system rejects file types that are not supported for CV upload.				
Steps	1. Attempt to upload a file that is not a PDF or DOCX in the CV upload flow.				
Expected	The system rejects the file and notifies the user that only PDF and DOCX formats are accepted.				
Date – Result	May 2026 — **PASS**				

TC-PER-06 – Save a Persona to Favourites

Test ID	TC-PER-07	Category	Functional	Severity	Major
Objective	This test case verifies that a logged-in user can save a persona to their favourites and retrieve it in future sessions.				
Steps	1. Create a persona and click the favourite icon for that persona. 2. Navigate away from the persona panel. 3. Return to the persona panel and check that the persona is still listed.				
Expected	The persona is stored and reappears without needing to be recreated.				
Date – Result	May 2026 — **PASS**				

TC-PER-07 – Delete a Saved Persona

Test ID	TC-PER-08	Category	Functional	Severity	Minor
Objective	This test case verifies that a user can permanently remove a saved persona from their favourites.				
Steps	1. Select a saved persona and click the delete option. 2. Confirm the deletion.				
Expected	The persona is removed and no longer appears in the grid.				
Date – Result	May 2026 — **PASS**				

TC-DEB-01 – Add Multiple Personas to the Council

Test ID	TC-DEB-01	Category	Functional	Severity	Critical
Objective	This test case verifies that a user can select multiple personas to form the debate council before submitting a question.				
Steps	1. From the persona grid, select three personas and add them to the debate council. 2. Verify that all three personas appear in the council tray. 3. Confirm that the debate submit button is active.				
Expected	All selected personas are shown in the council tray and the system is ready to start a debate.				
Date – Result	May 2026 — **PASS**				

TC-DEB-02 – Submit a Multi-Round Debate with Rebuttals

Test ID	TC-DEB-03	Category	Functional	Severity	Critical
Objective	This test case verifies that the system supports multi-round debates where personas can build on or challenge previous responses.				
Steps	1. Select three personas and enter a debatable question. 2. Set the round count to 2 and submit. 3. Observe Round 1 responses, then wait for Round 2 to complete.				
Expected	Round 1 shows initial positions. Round 2 shows rebuttals. The judge synthesizes all rounds into a final consensus.				
Date – Result	May 2026 — **PASS**				

TC-DEB-03 – Interface Stays Responsive While Waiting for Responses

Test ID	TC-DEB-04	Category	Functional	Severity	Critical
Objective	This test case verifies that the frontend does not freeze while the backend processes persona responses.				
Steps	1. Select four personas and submit a debate. 2. While waiting for responses, scroll the page, click on persona cards, and interact with the interface. 3. Observe whether the page remains usable throughout.				
Expected	The page remains fully interactive while waiting. Persona responses appear as soon as each one is ready, without blocking the others.				
Date – Result	May 2026 — **PASS**				

TC-DEB-04 – Personas Reason from Their Own Domain

Test ID	TC-DEB-05	Category	Functional	Severity	Critical
Objective	This test case verifies that each persona's response reflects its stated background and does not replicate another persona's perspective.				
Steps	1. Add a Security Specialist and a Performance Engineer persona. 2. Ask both the same question about enabling verbose logging in production. 3. Compare the two responses.				
Expected	The security persona focuses on audit trails and risk. The performance persona focuses on system overhead. The responses are clearly different in focus.				
Date – Result	May 2026 — **PASS**				

TC-DEB-05 – Judge Consensus is Generated After All Personas Respond

Test ID	TC-DEB-06	Category	Functional	Severity	Critical
Objective	This test case verifies that after all personas complete their responses, the system automatically generates a synthesized judge consensus.				
Steps	1. Complete a debate with at least two personas. 2. Wait for all persona cards to show as complete. 3. Locate the judge block and review its summary and reasoning.				
Expected	The judge produces a consensus that references and synthesizes the persona responses. Both the summary and the reasoning are present and non-empty.				
Date – Result	May 2026 — **PASS**				

TC-DEB-06 – Quality Scores Are Displayed for Each Persona

Test ID	TC-DEB-07	Category	Functional	Severity	Major
Objective	This test case verifies that the system shows relevance and reasoning quality scores for each persona after a debate.				
Steps	1. Complete a debate with three personas. 2. Locate the quality scores section in the results view.				
Expected	Each persona is assigned two scores: one for topic relevance and one for reasoning quality. All values are between 0 and 9.				
Date – Result	May 2026 — **PASS**				

TC-DEB-07 – Single-Persona Mode Returns a Direct Result

Test ID	TC-DEB-08	Category	Functional	Severity	Major
Objective	This test case verifies that when only one persona is used, the system returns that persona's answer directly as the consensus without running a separate judge call.				
Steps	1. Add exactly one persona to the council. 2. Submit a question and review the result.				
Expected	The persona's answer is returned as the consensus. A note indicates that single-persona mode was used.				
Date – Result	May 2026 — **PASS**				

TC-DEB-08 – Exceeding the Persona Limit Is Rejected

Test ID	TC-DEB-09	Category	Functional	Severity	Major
Objective	This test case verifies that the system enforces an upper limit on the number of personas that can be used in a single debate.				
Steps	1. Attempt to submit a debate with more personas than the maximum allowed.				
Expected	The request is rejected and the user is informed of the maximum allowed persona count.				
Date – Result	May 2026 — **PASS**				

TC-DEB-09 – Uploaded Document Is Used as Evidence

Test ID	TC-DEB-10	Category	Functional	Severity	Major
Objective	This test case verifies that a document attached to a debate question is processed and its content is made available to personas as evidence.				
Steps	1. Attach a document file to the debate question. 2. Submit the debate and review the persona responses.				
Expected	The system extracts the document content and provides it to the personas. Their responses reference or acknowledge the document material.				
Date – Result	May 2026 — **PASS**				

TC-DEB-10 – Empty Question Cannot Be Submitted

Test ID	TC-DEB-11	Category	Functional	Severity	Major
Objective	This test case verifies that the system prevents a debate from being submitted when the question field is empty.				
Steps	1. Select a persona but leave the question field empty. 2. Attempt to submit the debate.				
Expected	The submission is blocked. The user is informed that a question is required.				
Date – Result	May 2026 — **PASS**				

TC-DEB-11 – Debate Results Can Be Exported

Test ID	TC-DEB-12	Category	Functional	Severity	Minor
Objective	This test case verifies that a user can export the results of a completed debate to a readable format.				
Steps	1. Complete a debate and click the export option. 2. Review the exported content.				
Expected	The export includes the question, each persona's response per round, and the judge's summary and reasoning.				
Date – Result	May 2026 — **PASS**				

TC-SES-01 – Debate Is Saved to a Session

Test ID	TC-SES-01	Category	Functional	Severity	Critical
Objective	This test case verifies that when a logged-in user submits a debate within a session, the question, responses, and result are all persisted.				
Steps	1. Create a new session from the Session Sidebar. 2. Select personas, type a question, and submit the debate within that session. 3. Verify the session appears in the sidebar.				
Expected	The debate content is saved under the session. The session appears in the sidebar with the correct information.				
Date – Result	May 2026 — **PASS**				

TC-SES-02 – Session History Is Restored After Re-login

Test ID	TC-SES-02	Category	Functional	Severity	Critical
Objective	This test case verifies that a user can close and reopen the application and find their previous session fully restored.				
Steps	1. Complete a debate in a named session. 2. Close the browser and log back in. 3. Select the same session from the sidebar.				
Expected	The full conversation history is restored, including the question, all persona responses by round, and the judge consensus.				
Date – Result	May 2026 — **PASS**				

TC-SES-03 – Previous Context Is Available in Follow-up Questions

Test ID	TC-SES-03	Category	Functional	Severity	Major
Objective	This test case verifies that when a user asks a follow-up question in the same session, the system includes prior conversation context.				
Steps	1. Submit at least three debate questions in the same session. 2. On the third question, check whether the responses reference or acknowledge earlier exchanges.				
Expected	The system includes a summary of prior exchanges when generating new responses. Personas and the judge can refer to earlier context.				
Date – Result	May 2026 — **PASS**				

TC-SES-04 – Session Can Be Renamed

Test ID	TC-SES-04	Category	Functional	Severity	Major
Objective	This test case verifies that a user can change the name of an existing session.				
Steps	1. Click the edit option on a session name in the sidebar. 2. Enter a new name and confirm. 3. Verify the updated name is displayed.				
Expected	The session name is updated and appears correctly in the sidebar.				
Date – Result	May 2026 — **PASS**				

TC-SES-05 – Session Can Be Deleted

Test ID	TC-SES-05	Category	Functional	Severity	Major
Objective	This test case verifies that a user can permanently delete a session and all its associated messages.				
Steps	1. Select a session from the sidebar and choose to delete it. 2. Confirm the deletion.				
Expected	The session and all its messages are removed. The session no longer appears in the sidebar.				
Date – Result	May 2026 — **PASS**				

TC-SES-06 – Session Data Requires Authentication to Access

Test ID	TC-SES-06	Category	Functional	Severity	Minor
Objective	This test case verifies that session data is only accessible to authenticated users and cannot be accessed without a valid session.				
Steps	1. Without logging in, attempt to access the session list or any session's messages.				
Expected	Access is denied and the user is redirected to the login page or receives an authentication error.				
Date – Result	May 2026 — **PASS**				

TC-SEC-01 – Users Cannot Access Each Other's Sessions

Test ID	TC-SEC-01	Category	Security / Integration	Severity	Critical
Objective	This test case verifies that the system enforces user-level isolation so that one user cannot read, modify, or delete another user's sessions.				
Steps	1. Log in as User A and note the identifier of one of their sessions. 2. Log in as User B and attempt to view, modify, or delete User A's session using that identifier.				
Expected	Access is denied for all operations. Each user can only interact with their own data.				
Date – Result	May 2026 — **PASS**				

TC-SEC-02 – Passwords Are Stored Securely

Test ID	TC-SEC-02	Category	Security / Integration	Severity	Critical
Objective	This test case verifies that user passwords are never stored in a readable format in the database.				
Steps	1. Register a new account with a known password. 2. Inspect the password value stored in the database.				
Expected	The password is stored as a hash. The original password text is not visible in any database record.				
Date – Result	May 2026 — **PASS**				

TC-SEC-03 – Malicious Input Does Not Compromise the Database

Test ID	TC-SEC-03	Category	Security / Integration	Severity	Critical
Objective	This test case verifies that the system safely handles input that contains database injection patterns without altering or damaging stored data.				
Steps	1. Submit a debate with a question that contains common injection patterns. 2. Enter injection patterns in the persona name and description fields as well. 3. Verify that all database records remain intact.				
Expected	All injected text is treated as plain input. No database tables or records are affected.				
Date – Result	May 2026 — **PASS**				

TC-SEC-04 – Script Tags in User Input Are Not Executed

Test ID	TC-SEC-04	Category	Security / Integration	Severity	Critical
Objective	This test case verifies that content entered by a user that contains script tags is displayed as text and never executed in the browser.				
Steps	1. Create a persona with a description that contains a script tag. 2. Save and view the persona. Also use it in a debate and review the results.				
Expected	The script tag is displayed as visible text. No scripts are executed and no unexpected behavior occurs.				
Date – Result	May 2026 — **PASS**				

TC-SEC-05 – Expired Verification Codes Are Rejected

Test ID	TC-SEC-05	Category	Security / Integration	Severity	Major
Objective	This test case verifies that the system does not accept verification codes that have passed their expiry time.				
Steps	1. Register a new account and receive a verification code. 2. Allow the code to expire, then attempt to verify using it.				
Expected	The system rejects the expired code and instructs the user to request a new one.				
Date – Result	May 2026 — **PASS**				

TC-SEC-06 – Requests from Unlisted Origins Are Blocked

Test ID	TC-SEC-06	Category	Security / Integration	Severity	Major
Objective	This test case verifies that the cross-origin policy prevents the API from responding to requests from origins that are not in the allowed list.				
Steps	1. Attempt to make an API request from a browser origin that is not configured as allowed.				
Expected	The request is blocked by the cross-origin policy. No data is returned to the requesting origin.				
Date – Result	May 2026 — **PASS**				

TC-SEC-07 – Anonymous Users Have a Daily Debate Limit

Test ID	TC-SEC-07	Category	Security / Integration	Severity	Major
Objective	This test case verifies that unauthenticated users cannot submit an unlimited number of debates per day.				
Steps	1. Without logging in, submit the maximum number of debates allowed for anonymous users. 2. Attempt to submit one additional debate.				
Expected	The allowed debates succeed. The additional request is rejected with a message stating the daily limit has been reached.				
Date – Result	May 2026 — **PASS**				

TC-SEC-08 – Registered Users Have a Daily Debate Limit

Test ID	TC-SEC-08	Category	Security / Integration	Severity	Major
Objective	This test case verifies that authenticated regular users cannot exceed the configured daily debate allowance.				
Steps	1. Log in as a regular user and submit the maximum allowed number of debates. 2. Attempt to submit one additional debate.				
Expected	The allowed debates succeed. The additional request is rejected with a daily limit message.				
Date – Result	May 2026 — **PASS**				

TC-SEC-09 – Admin Pages Are Not Accessible to Regular Users

Test ID	TC-SEC-09	Category	Security / Integration	Severity	Minor
Objective	This test case verifies that the admin statistics page is only accessible to users with admin privileges.				
Steps	1. Log in as a regular user and attempt to access the admin statistics page. 2. Log in as an admin user and access the same page.				
Expected	The regular user is denied access. The admin user can view the statistics page.				
Date – Result	May 2026 — **PASS**				

TC-LLM-01 – System Produces Results Without an LLM API Key

Test ID	TC-LLM-01	Category	Functional	Severity	Critical
Objective	This test case verifies that the system gracefully handles the absence of a configured LLM API key by returning simulated responses.				
Steps	1. Remove all LLM API keys from the environment configuration. 2. Submit a debate and observe the response.				
Expected	The system returns structurally valid simulated responses. No crash or unhandled error occurs.				
Date – Result	May 2026 — **PASS**				

TC-LLM-02 – Persona Calls Run in Parallel

Test ID	TC-LLM-02	Category	Functional	Severity	Critical
Objective	This test case verifies that all persona LLM calls for a round are dispatched simultaneously rather than one after another.				
Steps	1. Add five personas and submit a debate. 2. Compare the total response time to a single-persona debate.				
Expected	The total wait time is approximately equal to the duration of one call, not five times longer.				
Date – Result	May 2026 — **PASS**				

TC-LLM-03 – Topic Relevance Scores Reflect Persona Background

Test ID	TC-LLM-03	Category	Functional	Severity	Major
Objective	This test case verifies that the pre-debate scoring correctly identifies which personas are more relevant to the given question.				
Steps	1. Add one persona clearly relevant to the question and one with an unrelated background. 2. Complete the debate and compare their topic relevance scores.				
Expected	The relevant persona receives a noticeably higher score. All scores are between 0 and 9.				
Date – Result	May 2026 — **PASS**				

TC-LLM-04 – Reasoning Quality Scores Are Assigned After the Debate

Test ID	TC-LLM-04	Category	Functional	Severity	Major
Objective	This test case verifies that each persona receives a reasoning quality score based on the strength of their argument.				
Steps	1. Complete a debate with two to three personas. 2. Check that each persona has a reasoning quality score and an accompanying rationale.				
Expected	Every persona has a score between 0 and 9. The rationale briefly explains the basis for the score.				
Date – Result	May 2026 — **PASS**				

TC-LLM-05 – Judge Responses Are More Consistent Than Persona Responses

Test ID	TC-LLM-05	Category	Functional	Severity	Major
Objective	This test case verifies that the judge model produces more stable output across repeated runs than the individual persona models.				
Steps	1. Submit the identical question with the same personas three separate times. 2. Compare the judge summaries and the persona answers across all three runs.				
Expected	The judge summaries are more consistent than the persona answers, reflecting the lower temperature applied to the judge model.				
Date – Result	May 2026 — **PASS**				

TC-LLM-06 – Researcher Persona Response Reflects Publication Background

Test ID	TC-LLM-06	Category	Functional	Severity	Minor
Objective	This test case verifies that when a researcher persona is used, its response reflects themes found in the researcher's actual publications.				
Steps	1. Create a persona using the researcher pipeline. 2. Submit a question within the researcher's domain of expertise. 3. Review the persona response.				
Expected	The response references themes or methods consistent with the researcher's publication background.				
Date – Result	May 2026 — **PASS**				

TC-LLM-07 – LLM Failure Returns a Readable Error Message

Test ID	TC-LLM-07	Category	Functional	Severity	Minor
Objective	This test case verifies that when the LLM service cannot be reached, the system returns a clear error rather than crashing.				
Steps	1. Configure an invalid API key in the environment. 2. Submit a debate and check the response.				
Expected	The system catches the failure and returns a descriptive error message. The application does not crash.				
Date – Result	May 2026 — **PASS**				

TC-DEP-01 – Health Check Confirms the Backend Is Running

Test ID	TC-DEP-01	Category	Functional	Severity	Major
Objective	This test case verifies that the health check endpoint confirms that the backend service is operational.				
Steps	1. Send a request to the health check endpoint. 2. Check the response.				
Expected	The endpoint returns a success status indicating the backend is running normally.				
Date – Result	May 2026 — **PASS**				

TC-DEP-02 – Database Tables Are Created on First Launch

Test ID	TC-DEP-02	Category	Functional	Severity	Major
Objective	This test case verifies that the system creates all required database tables automatically on first startup without manual intervention.				
Steps	1. Start the backend for the first time against an empty database. 2. Check that all expected tables have been created after startup.				
Expected	All required tables are present. No manual database setup was required.				
Date – Result	May 2026 — **PASS**				

TC-DEP-03 – Full Application Stack Starts with Docker Compose

Test ID	TC-DEP-03	Category	Functional	Severity	Major
Objective	This test case verifies that all three services start correctly using Docker Compose and the application is accessible in a browser.				
Steps	1. Add the required API keys to the environment configuration file. 2. Start all services using Docker Compose. 3. Open the application in a web browser.				
Expected	All services start successfully and the Consensia interface loads correctly in the browser.				
Date – Result	May 2026 — **PASS**				

TC-DEP-04 – Frontend Requests Reach the Backend Without Cross-Origin Errors

Test ID	TC-DEP-04	Category	Functional	Severity	Minor
Objective	This test case verifies that the proxy configuration correctly routes API requests from the frontend to the backend without triggering cross-origin errors.				
Steps	1. Start the full application stack. 2. Submit a debate and observe the network requests in the browser.				
Expected	Requests from the frontend reach the backend successfully. No cross-origin errors appear in the browser.				
Date – Result	May 2026 — **PASS**				

5.3 Non-Functional Test Cases

The following 15 test cases verify the performance, security, reliability, and usability properties of the system.

TC-NF-01 – Page Stays Interactive While Waiting for a Debate

Test ID	TC-NF-01	Category	Performance / Non-Functional	Severity	Critical
Objective	This test case verifies that the frontend remains fully usable while the backend is processing a debate request.				
Steps	1. Submit a debate with five personas. 2. While waiting for responses, scroll, click, and type in the interface.				
Expected	The page remains fully responsive. No parts of the interface freeze during the wait.				
Date – Result	May 2026 — **PASS**				

TC-NF-02 – Persona Responses Are Generated Within an Acceptable Time

Test ID	TC-NF-02	Category	Performance / Non-Functional	Severity	Critical
Objective	This test case verifies that the average response time for a single persona call is within an acceptable range under normal conditions.				
Steps	1. Submit a single-persona debate with a short question five times. 2. Record the response time for each run and calculate the average.				
Expected	The average response time does not exceed ten seconds under normal conditions.				
Date – Result	May 2026 — **PASS**				

TC-NF-03 – Non-Debate Endpoints Respond Quickly

Test ID	TC-NF-03	Category	Performance / Non-Functional	Severity	Major
Objective	This test case verifies that endpoints that do not involve LLM calls respond within a short time frame.				
Steps	1. Request the session list, persona favourites list, and health check endpoints. 2. Measure the response time for each.				
Expected	All three endpoints respond within 200 milliseconds under normal load.				
Date – Result	May 2026 — **PASS**				

TC-NF-04 – System Handles Multiple Concurrent Users

Test ID	TC-NF-04	Category	Performance / Non-Functional	Severity	Major
Objective	This test case verifies that the system can serve multiple simultaneous users without errors or data mixing.				
Steps	1. Send twenty simultaneous requests to retrieve the session list from different authenticated accounts. 2. Verify that all requests return the correct data for each respective user.				
Expected	All requests return correct results. No errors occur and no user receives another user's data.				
Date – Result	May 2026 — **PASS**				

TC-NF-05 – Oversized Input Is Truncated Without Errors

Test ID	TC-NF-05	Category	Performance / Non-Functional	Severity	Major
Objective	This test case verifies that when input text exceeds the system's configured limit, it is truncated safely and processing continues.				
Steps	1. Upload a CV that contains more text than the system's processing limit. 2. Check that a persona is still generated.				
Expected	The text is truncated at the limit and a valid persona is generated. No error is returned.				
Date – Result	May 2026 — **PASS**				

TC-NF-06 – Application Loads Within a Reasonable Time

Test ID	TC-NF-06	Category	Performance / Non-Functional	Severity	Minor
Objective	This test case verifies that the application becomes fully usable within an acceptable time after opening it in a browser.				
Steps	1. Clear the browser cache and open the application. 2. Measure the time until the page is fully interactive.				
Expected	The page becomes interactive within a few seconds on a standard connection.				
Date – Result	May 2026 — **PASS**				

TC-NF-07 – A Single Failing Persona Does Not Cancel the Whole Debate

Test ID	TC-NF-07	Category	Reliability / Non-Functional	Severity	Critical
Objective	This test case verifies that the system tolerates individual persona failures and continues processing with the remaining personas.				
Steps	1. Configure a scenario where one persona's generation will fail. 2. Submit a debate with four personas and observe the result.				
Expected	The three successful personas complete normally. The judge produces a consensus from the available answers. The failed persona shows an error state.				
Date – Result	May 2026 — **PASS**				

TC-NF-08 – Raw CV Content Is Not Stored in the Database

Test ID	TC-NF-08	Category	Security / Non-Functional	Severity	Major
Objective	This test case verifies that the system does not persist the raw text of uploaded CV files, in line with data minimization principles.				
Steps	1. Upload a CV with clearly identifiable personal details. 2. After the persona is generated, inspect the stored database records.				
Expected	Only the generated persona fields are stored. The original CV text is not present in any database record.				
Date – Result	May 2026 — **PASS**				

TC-NF-09 – Researcher Search Without API Key Returns a Clear Message

Test ID	TC-NF-09	Category	Security / Non-Functional	Severity	Major
Objective	This test case verifies that when the researcher search service is not configured, the system responds with an informative error rather than crashing.				
Steps	1. Remove the researcher search API key from the configuration. 2. Attempt to create a persona through the researcher search flow.				
Expected	The system returns a clear message stating that the service is not configured. The application continues to function otherwise.				
Date – Result	May 2026 — **PASS**				

TC-NF-10 – Application Works on Major Browsers

Test ID	TC-NF-10	Category	Usability / Non-Functional	Severity	Major
Objective	This test case verifies that the application functions correctly across the three most widely used desktop browsers.				
Steps	1. Open the application in Chrome, Firefox, and Edge. 2. Complete the full user flow in each: register, create a persona, run a debate, and view the results.				
Expected	All features function correctly in all three browsers without visual or functional issues.				
Date – Result	May 2026 — **PASS**				

TC-NF-11 – Interface Is Usable on Small Screen Sizes

Test ID	TC-NF-11	Category	Usability / Non-Functional	Severity	Minor
Objective	This test case verifies that the application layout adapts to smaller screen widths without losing usability.				
Steps	1. Open the application in a browser set to a narrow viewport width. 2. Navigate through the persona panel, question input, and results view.				
Expected	All interface elements remain accessible and readable. No critical content is hidden or cut off.				
Date – Result	May 2026 — **PASS**				

TC-NF-12 – Oversized CV Is Processed Without an Error

Test ID	TC-NF-12	Category	Reliability / Non-Functional	Severity	Major
Objective	This test case verifies that uploading a CV larger than the system limit does not cause an error, and a persona is still generated.				
Steps	1. Upload a CV that produces more extractable text than the configured limit. 2. Observe whether a persona is generated.				
Expected	The text is truncated at the limit and a valid persona is returned. No error is thrown.				
Date – Result	May 2026 — **PASS**				

TC-NF-13 – Non-Document Files Cannot Be Uploaded as CVs

Test ID	TC-NF-13	Category	Security / Non-Functional	Severity	Major
Objective	This test case verifies that the system rejects files that are not valid documents when submitted through the CV upload flow.				
Steps	1. Attempt to upload a file of an unsupported type through the CV upload interface.				
Expected	The file is rejected with an appropriate error message. No part of the file is processed or stored.				
Date – Result	May 2026 — **PASS**				

TC-NF-14 – Deleting a User Removes All Their Data

Test ID	TC-NF-14	Category	Reliability / Non-Functional	Severity	Major
Objective	This test case verifies that removing a user account results in the deletion of all data associated with that account.				
Steps	1. Create a user account with sessions, messages, and saved personas. 2. Delete the user account. 3. Check the database for any records belonging to the deleted user.				
Expected	All data associated with the account is removed. No orphaned records remain.				
Date – Result	May 2026 — **PASS**				

TC-NF-15 – Data Is Preserved After a Backend Restart

Test ID	TC-NF-15	Category	Reliability / Non-Functional	Severity	Minor
Objective	This test case verifies that restarting the backend service does not result in loss of previously stored data.				
Steps	1. Run a debate and save it to a session. 2. Restart the backend service. 3. Log back in and verify that the session and its content are still accessible.				
Expected	All data is preserved after the restart. The application functions normally after the service comes back online.				
Date – Result	May 2026 — **PASS**				

---

6. Maintenance Plan

6.1 Bug Fixes

Issues will be tracked through the project repository. Critical bugs affecting system stability or user data are prioritized for immediate resolution. Less critical issues are addressed in scheduled updates. Each fix should be accompanied by a corresponding test case to prevent recurrence.

6.2 LLM Provider Updates

The LLM service is designed to be provider-agnostic. Adding or changing a provider only requires modifications to the service layer. Model names are configurable through environment variables, allowing model upgrades without code changes.

6.3 Database Schema Changes

Any schema change must be introduced as a new migration file. Migrations are applied automatically on startup, ensuring that deployments to new environments require no manual database configuration.

6.4 Planned Improvements

Based on the current state of the system, the following enhancements are planned for future development:

- Streaming responses: display persona answers as they are generated rather than waiting for the full response.
- Persona interaction: allow personas to read and respond to each other's answers within the same debate.
- Confidence scoring: add an indicator showing how much the personas agreed or disagreed, giving users a sense of the certainty behind the consensus.

- Production deployment: replace the development server with a production-ready web server and add HTTPS support.
- Researcher caching: avoid re-scraping the same researcher on repeated requests by storing their profile locally.

6.5 Regular Maintenance

Periodic tasks include rotating authentication secrets and API keys, checking for security updates in dependencies and Docker base images, reviewing database records for cleanup, and running dependency security audits.

---

7. Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

7.1.1 Constraints

API Cost: The reliance on external LLM services with usage-based pricing was a constraint throughout the project. Prompt length limits, the choice of a smaller model as the primary provider, and the maximum persona count per debate were all decided in part to manage operational costs.

Response Latency: A multi-persona, multi-round debate involves many LLM calls. Parallel execution reduces this significantly, but total response time still grows with the number of rounds. This was an accepted trade-off given the richer output quality.

External Service Dependencies: Both the LLM providers and the researcher scraping service impose rate limits and may change their interfaces over time. Bundled researcher snapshots and a simulated fallback mode provide partial resilience against these dependencies.

Infrastructure: The full Docker Compose stack requires a reasonably capable machine. The current configuration is suited to development and small-scale deployment. Production hardening is identified as future work.

7.1.2 Standards

REST API Design: All backend endpoints follow REST conventions using standard HTTP methods and JSON request and response bodies [13].

JSON Format (RFC 8259): All data exchanged between system components uses JSON. Input and output schemas are validated at runtime [13].

GDPR Compliance: CV files are processed in memory and not stored. Only the generated persona description is persisted, following the data minimization principle [12].

Security Standards: Password hashing, token-based authentication, cross-origin protection, and rate limiting were implemented in accordance with accepted web application security practices [12].

Accessibility (WCAG 2.1): Basic accessibility guidelines were followed for the frontend theme, including text contrast and keyboard navigation in form elements [11].

The table below summarizes the effect level of each factor on the engineering design of Consensia, on a scale from 0 (no effect) to 10 (maximum effect):

Factor	Discussion	Effect (0–10)
Public Health & Safety	Consensia has no direct impact on physical safety. All outputs are labelled as AI-generated and the system is designed to assist decision making, not replace it.	2
Ethics	Multiple personas reduce single-perspective bias. Researcher personas are based on public data and labelled as AI simulations. All reasoning steps are transparent and visible to the user.	7
Security & Privacy	Authentication, password hashing, session-level access control, rate limiting, and cross-origin policies were all implemented in direct response to security requirements.	9
Economic Factors	LLM API costs influenced the choice of provider, the prompt length limits, and the maximum persona count per debate.	8
Environmental Factors	A smaller and faster model was chosen as the primary provider to reduce inference energy use. Parallel execution reduces idle processing time between calls.	4

Factor	Discussion	Effect (0–10)
Social & Cultural	Multi-perspective reasoning supports more balanced outcomes. The interface is designed to be accessible to users with varying levels of AI familiarity.	5
Legal Compliance (GDPR)	CV files are processed in memory and not stored. Only the generated persona description is persisted. This follows the data minimization principle.	8
Maintainability	The backend is divided into separate modules. All configuration is managed through environment variables. Schema changes are tracked through versioned migration files.	9
Scalability	The stateless authentication model supports horizontal scaling. Parallel async processing handles concurrent requests efficiently.	7
Accessibility (WCAG)	Basic accessibility principles were applied to the theme including text contrast and keyboard navigation. A full accessibility audit was outside the scope of this project.	4

7.2 Ethics and Professional Responsibilities

During the course of this project, the team identified and addressed several ethical and professional responsibilities.

Transparency: All outputs are labelled as AI-generated. The system is positioned as a decision-support tool and not as an authoritative source. The judge's reasoning is always displayed so users can evaluate the result rather than accepting it without scrutiny.

Persona ethics: Researcher personas are constructed from publicly available academic publication data. The system does not impersonate real individuals; it uses their published research to simulate how they might reason about a software engineering problem, and this is communicated to users.

Bias awareness: The multi-persona approach surfaces diverse viewpoints, which reduces but does not eliminate bias [1]. If all available personas share the same underlying biases, the consensus will reflect those. This limitation is acknowledged in the project documentation.

Data privacy: CV uploads are voluntary and users upload their own files. Raw CV content is never stored, aligning with data minimization principles [11].

Academic integrity: All libraries, APIs, and research papers used are properly cited. The report clearly describes how existing systems such as AutoGen and CrewAI relate to and differ from Consensia.

7.3 Teamwork Details

7.3.1 Contributing and Functioning Effectively

The workload was divided equally among all of us. However, we didn't work on all sections of the project. Reports are prepared with every team members' contributions from the team members. Development was managed through GitHub with pull requests and code reviews required before merging any branch.

Amirhossein Ahani

Engineered the multi-agent orchestration engine, including the judge consensus logic, dynamic proficiency weighting, and heuristic calibration signals. He developed the multimodal pipeline for image and PDF analysis via Azure OpenAI and implemented the bounded memory system using rolling session summaries to optimize token usage. Additionally, he built the production-ready infrastructure, including the 3-tier daily quota system for different user types and the secure 3-step password recovery flow.

Ahmed Hatem Haikal

Created the dataset that formed the basis of the study and contributed to the development of the system's frontend and backend components. He also worked on the database design and implementation, supporting the persistence of user accounts, debate sessions, message history, persona data, and other system records.

İrfan Hakan Karakoç

Introduced scraping pipeline and personification, improved it to its final state with increased accuracy of the created personas. Worked mostly on the research part of the project.

Mehmet Hakan Yavuz

Worked on the scraping method's staged pipeline and improved the accuracy of the created personas by introducing/researching prompt engineering methods, conducting a literature review in the software field.

Türker Köken

Handled bug fixes, and implemented frontend layout changes. Documented team discussions and meetings (additional notes, reports, presentations, recordings). Communicated literature review findings with the team.

7.3.2 Collaborative and Inclusive Environment

The team maintained regular communication through a shared messaging channel on WhatsApp and held weekly scheduled meetings. Weekly meetings were also held with a doctoral researcher from the supervisor's research group whose work on tangled commits is directly related to the project. This collaboration shaped several technical decisions.

Technical disagreements were resolved through structured discussion where options were evaluated against the project's stated design goals. Decisions were documented with their rationale.

7.3.3 Taking Lead Role and Sharing Leadership

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.

7.3.4 Meeting Objectives

Milestone	Status	Notes
System analysis and requirements	Complete	Requirements documented and design goals defined.
Backend API (FastAPI + PostgreSQL)	Complete	All endpoints implemented with 13 database migrations.
Basic LLM consensus endpoint	Complete	Single-turn consensus working with Gemini and OpenAI.
Multi-round debate with parallel calls	Complete	Up to 3 rounds with concurrent calls and session context.
Two-pass QA scoring	Complete	Topic relevance and reasoning quality scoring integrated.

Milestone	Status	Notes
Persona creation — manual and CV paths	Complete	PDF and DOCX extraction working correctly.
Researcher persona pipeline	Complete	Multi-stage pipeline with bundled snapshots for offline use.
Authentication (email and Google OAuth)	Complete	Full auth flow including verification and password reset.
Session persistence and message history	Complete	Full session CRUD with rolling summary.
Rate limiting and admin panel	Complete	Daily limits enforced for anonymous and registered users.
Docker Compose deployment	Complete	Three-service stack with health checks.
Functional and non-functional testing	Complete	50 functional test cases and 15 non-functional test cases were completed, for a total of 65 tests. .
Streaming responses	Planned	May be implemented as a future improvement.

7.4 New Knowledge Acquired and Applied

This project required the team to learn and apply knowledge across several areas not fully covered in coursework.

Asynchronous Python: The concurrent execution of multiple LLM calls required practical understanding of Python's async/await model and parallel task dispatch, which was applied directly in the debate orchestration logic [8, 14].

LLM Prompt Engineering: Designing effective prompts for persona role-play, calibration scoring, and structured output required significant iteration [7]. Key lessons included the effect of temperature on output consistency and the importance of explicit format instructions [15].

Database Migrations: Managing schema changes through versioned migration files was a new practice for most team members. Handling a branched migration history and writing data migrations were specific challenges encountered and resolved during development.

Researcher Data Pipeline: Designing a multi-stage processing pipeline that chunked and aggregated publication data required understanding of context window constraints and structured output parsing from language models.

Authentication Implementation: Implementing the full email-based registration flow and Google OAuth integration provided practical experience with security patterns that were previously only known in theory.

Container Networking: Understanding how services communicate within Docker Compose and how to configure proxy routing between the frontend and backend was new but essential for the deployment setup [6, 16, 17, 18, 19].

---

8. Conclusion and Future Work

Consensia achieves its primary goal of providing a structured multi-perspective reasoning platform for software engineering decisions. The system allows users to create personas from multiple sources, conduct multi-round debates with concurrent language model calls, receive calibrated judge consensus with traceable reasoning, and revisit full conversation histories through persistent sessions.

The two-pass quality scoring system is one of the more significant contributions of this project. By evaluating both how relevant a persona is to the question and how well they argued their position, the judge has a principled basis for weighting contributions rather than treating all responses equally.

Regarding the research questions the project set out to address: language models can reasonably simulate distinct personas when given well-structured background descriptions. A judge model can synthesize multiple perspectives into a consistent and traceable consensus. Including actual researcher publication data as persona background material produces more grounded and specific responses. The main limitation is that biases in the underlying language models are not eliminated by the multi-persona structure — they are diversified, which is an improvement but not a guarantee of correctness.

Future work will focus on streaming responses to reduce perceived latency, inter-persona interaction for more structured debate, confidence indicators in the judge output, and an empirical evaluation of the system on real tangled commit classification tasks.

---

9. Glossary

AI	Artificial Intelligence - systems designed to perform tasks that typically require human reasoning.
LLM	Large Language Model — a machine learning model trained on large text datasets to generate natural language.
API	Application Programming Interface - a defined way for software components to communicate.
Alembic	A migration tool for SQLAlchemy that tracks database schema changes through versioned scripts.
Argon2id	A password hashing algorithm widely recommended for secure credential storage.
Docker Compose	A tool for defining and running multi-container applications through a single configuration file.
FastAPI	A Python framework for building REST APIs with built-in request validation.
GDPR	General Data Protection Regulation - EU law governing the collection and processing of personal data.
Google Gemini	Google's family of large language models; Consensia uses Gemini 2.0 Flash as the primary provider.
JWT	JSON Web Token - a compact token format used to maintain authenticated sessions.
Judge LLM	The model session in Consensia that synthesizes persona answers into a weighted consensus.
Persona	An AI agent configured with a professional background to generate domain-specific responses.
PostgreSQL	An open-source relational database used for all persistent storage in Consensia.
QA Scoring	The two-pass evaluation: topic relevance before debate, reasoning quality after. Both scored 0-9.
React	A JavaScript library for building user interfaces from reusable components.
REST	An architectural style for web APIs using standard HTTP methods.
SerpAPI	A service for querying search engine results programmatically, used here for Google Scholar.
SQLAlchemy	A Python ORM for interacting with relational databases through Python objects.
Tangled Commit	A version control commit containing multiple unrelated changes, making it difficult to review.
Tailwind CSS	A utility-first CSS framework used to build the frontend interface.
Vite	A frontend build tool and development server used for the React application.

WCAG 2.1	Web Content Accessibility Guidelines - a standard for accessible web content.

10. References

[1] Y. Du et al., "Improving Reasoning with Multi-Agent Debate," ICML, 2024.

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

[3] PostgreSQL Global Development Group, "PostgreSQL 16 Documentation," 2024.

[4] SQLAlchemy, "SQLAlchemy 2.0 Documentation," Available: https://docs.sqlalchemy.org.

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

[6] R. Fielding and R. Taylor, "Architectural Styles and the Design of Network-Based Software Architectures," UC Irvine, 2000.

[7] L. Zheng et al., "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," NeurIPS, 2023.

[8] pwdlib, "Modern Password Hashing for Python," Available: https://pypi.org/project/pwdlib.

[9] Herbold, S., Trautsch, A., Ledel, B. *et al.* A fine-grained data set and analysis of tangling in bug fixing commits. *Empir Software Eng* **27**, 125 (2022). https://doi.org/10.1007/s10664-021-10083-5

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

[11] W3C, "Web Content Accessibility Guidelines (WCAG) 2.1," 2018.

[12] W. Stallings, Effective Cybersecurity. Addison-Wesley, 2018.

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

[14] J. Wang et al., "CrewAI: Coordinating Role-Playing Autonomous Agents," 2024.

[15] T. Brown et al., "Language Models are Few-Shot Learners," NeurIPS, 2020.

[16] Google, "Google Scholar," Available: https://scholar.google.com.

[17] SerpAPI, "SerpAPI Documentation," Available: https://serpapi.com.

[18] Google DeepMind, "Gemini API Documentation," Available: https://ai.google.dev.

[19] FastAPI, "FastAPI Documentation," Available: https://fastapi.tiangolo.com.

[20] OpenAI, "OpenAI Evals Framework," 2023.

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