Semblance

Technical Architecture Document

AI-Powered Synthetic Focus Group Research Platform

Version 1.0

February 2026

Table of Contents

CHAPTER 1

Executive Overview

System Purpose and Capabilities

Executive Overview

Purpose

Semblance is an AI-powered synthetic focus group research platform that enables researchers, product teams, and UX professionals to create detailed synthetic personas using large language models, organize them into focus groups, and conduct moderated or fully autonomous research sessions — all without recruiting real participants.

The platform supports multi-model AI integration (Google Gemini, OpenAI GPT-4.1 and GPT-5.2), real-time WebSocket communication for live session collaboration, and comprehensive analysis tools including sentiment analysis, theme extraction, and participation analytics.

Key Capabilities

- **AI Persona Generation** — Two-stage pipeline: basic demographic profiles then full personality expansion (OCEAN traits, goals, frustrations, motivations, scenarios, AI-synthesized biography).
- **Focus Group Simulation** — Manual moderation or fully autonomous AI-driven conversations with real-time WebSocket updates.
- **Multi-Model LLM Support** — Unified service abstracting Google Gemini and OpenAI models with retry logic and model-specific parameter handling.
- **Real-Time Collaboration** — Socket.IO room-based messaging for live session observation with event-driven UI updates.
- **Comprehensive Analysis** — AI-powered theme extraction, sentiment analysis, participation balance scoring, and exportable reports.
- **Enterprise Authentication** — Dual auth: local JWT credentials and Microsoft Entra ID (MSAL) OAuth.

Technology Summary

Layer	Technologies
Frontend	React 18, TypeScript, Vite, Tailwind CSS, shadcn-ui (Radix UI), React Router, TanStack Query, Socket.IO Client
Backend	Python, Quart (async Flask), Hypercorn ASGI, python-socketio, PyMongo
Database	MongoDB (4 collections: users, personas, focus_groups, folders)
AI / LLM	Google Gemini (gemini-3-pro-preview), OpenAI GPT-4.1, OpenAI GPT-5.2
Authentication	Custom JWT (HS256, 24h expiry), Microsoft MSAL (Entra ID)
Real-Time	Socket.IO (WebSocket with polling fallback)

System Architecture Overview

Semblance follows a three-tier architecture: a React single-page application communicates with a Python Quart backend through REST APIs and WebSocket connections. The backend orchestrates multiple LLM providers and persists all data in MongoDB.

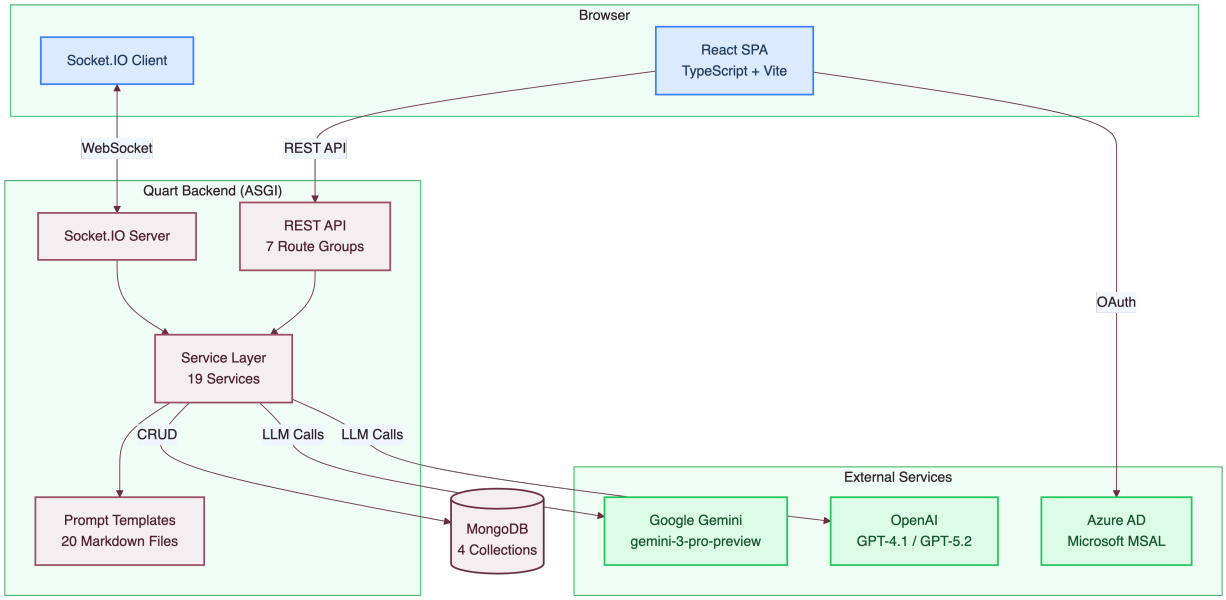

Figure 1.1 — High-Level System Architecture

CHAPTER 2

System Architecture

Deployment and Infrastructure

System Architecture

Three-Tier Architecture

The application is organized into three distinct tiers, each independently deployable:

- **Presentation Tier** — React SPA built with Vite, served as static assets. Handles all UI rendering, client-side routing, and WebSocket event dispatching.
- **Application Tier** — Quart (async Flask) application running under Hypercorn ASGI server. Hosts the REST API (7 blueprint groups), Socket.IO server, 19 business logic services, and a dedicated AI runner thread for autonomous conversations.
- **Data Tier** — MongoDB document database storing users, personas, focus groups, and folders. Accessed via PyMongo (sync) in route handlers and Motor (async) in the AI runner thread.

Deployment Topology

In production, the application is deployed at **ai-sandbox.oliver.solutions** behind an Nginx reverse proxy that routes requests to either the static frontend assets or the backend application server.

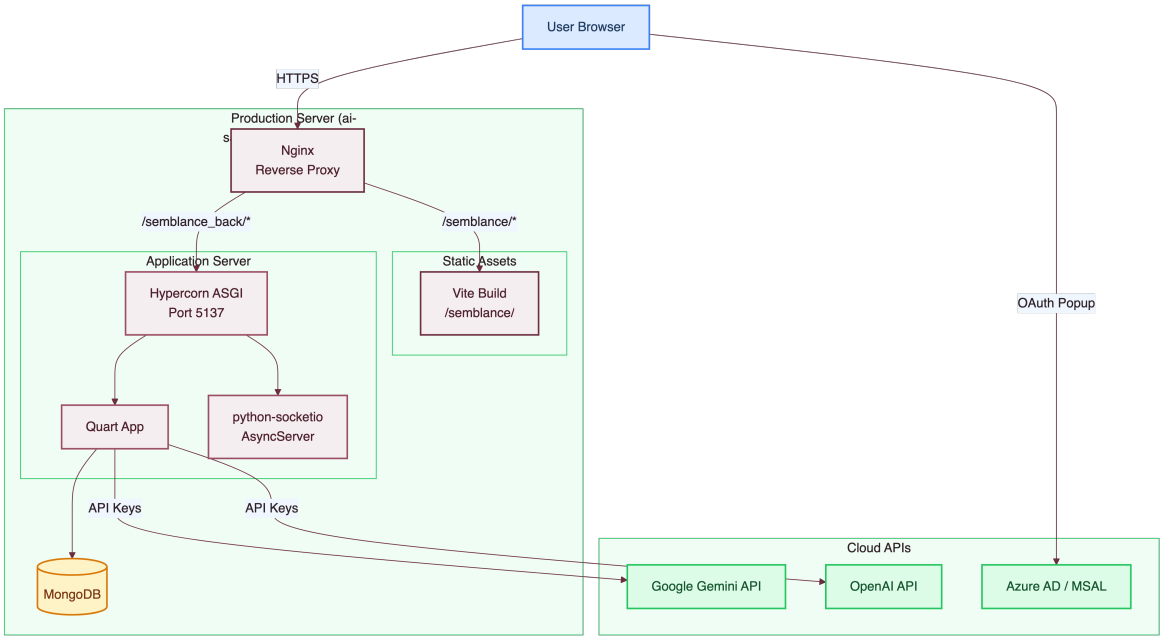

Figure 2.1 — Production Deployment Architecture

Environment Configuration

The application supports dual environments through Vite environment files. Configuration switches base paths, API URLs, WebSocket paths, and authentication redirects.

Setting	Development	Production
Base Path	/	/semblance/
API Base URL	/api	https://ai-sandbox.oliver.solutions/semblance_back/api
WebSocket Path	/socket.io/	/semblance_back/socket.io/
Frontend Port	5173 (Vite dev server)	Static assets via Nginx
Backend Port	5137 (Hypercorn)	5137 (proxied via Nginx)
MSAL Redirect	http://localhost:5173/	https://ai-sandbox.oliver.solutions/semblance

Application Factory Pattern

The backend uses an application factory pattern (**create_app()** in app/__init__.py) that initializes the Quart app, configures CORS, registers 7 route blueprints, sets up JWT authentication, initializes the WebSocket manager, and starts the AI runner service. Key configuration:

- **JWT Secret** — From JWT_SECRET_KEY environment variable
- **Token Expiry** — 86,400 seconds (24 hours)
- **Max Upload** — 16 MB
- **Request Timeout** — 300 seconds (5 minutes)
- **CORS** — allow_origin="*" for all methods

CHAPTER 3

3 Frontend Architecture

React SPA Structure and Patterns

Frontend Architecture

Technology Stack

The frontend is a React 18 single-page application built with TypeScript and Vite. UI components use shadcn-ui (Radix UI primitives) styled with Tailwind CSS. Data fetching uses TanStack Query with form handling via React Hook Form and Zod validation.

Provider Hierarchy

App.tsx wraps the entire application in a nested provider hierarchy. Each provider adds a layer of functionality accessible throughout the component tree:

Provider	Purpose	Key State
QueryClientProvider	TanStack Query data fetching and caching	Query cache, stale-while-revalidate
BrowserRouter	Client-side routing with dynamic base path	Route location, navigation
MsalProvider	Microsoft Azure AD authentication	MSAL instance, account info
AuthProvider	JWT token management and session persistence	user, token, isAuthenticated
WebSocketProvider	Singleton Socket.IO connection management	socketId, connection state
NavigationProvider	Navigation state and focus group context	previousRoute, focusGroupId, folderId

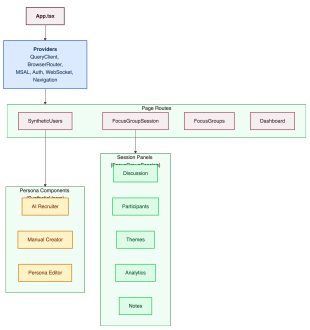

Figure 3.1 — Frontend Component Hierarchy

Route Structure

Path	Component	Auth	Description
/	Index	No	Landing page with platform overview
/login	Login	No	Authentication (local + Microsoft OAuth)
/synthetic-users	SyntheticUsers	Yes	Persona library and management
/synthetic-users/:id	PersonaProfile	Yes	Individual persona detail view
/focus-groups	FocusGroups	Yes	Focus group listing and creation
/focus-groups/:id	FocusGroupSession	Yes	Live session interface (multi-panel)
/dashboard	Dashboard	Yes	Analytics and research metrics

State Management Strategy

The application uses a layered state management approach rather than a single global store:

- **Global Persisted State** — AuthContext (JWT + user in localStorage), NavigationContext (route history in localStorage), TanStack Query cache (server data).
- **Component State** — React hooks for UI state (tabs, modals, filters), React Hook Form for form state, temporary editing data.
- **WebSocket State** — Connection status, real-time updates dispatched as window CustomEvents (ws:message_update, ws:ai_status_update, etc.).

Component Organization

Directory	Contents
src/components/ui/	Reusable shadcn-ui components (Button, Card, Dialog, Tabs, etc.) plus custom components (ProgressModal, MentionInput, SaveStatusIndicator)
src/components/focus-group-session/	25+ components for the session interface: DiscussionPanel, ParticipantPanel, ThemesPanel, AnalyticsPanel, ReasoningPanel, NotesPanel, AutonomousDashboard
src/components/persona/	Persona profile viewing and editing: PersonaProfile, PersonaEditor, PersonaPersonality, PersonaAttitudinalProfile, PersonaScenarios
src/components/dashboard/	Dashboard analytics: StatCard, OverviewTab, UsersTab, FocusGroupsTab
src/components/auth/	Authentication: MsalProvider (Azure AD setup)
src/hooks/	Custom hooks: useWebSocket, useCancellableGeneration, usePersonaFiltering, useFocusGroupAutoSave, useFolderManagement
src/services/	WebSocket singleton service with event dispatching via window CustomEvents
src/types/	TypeScript type definitions: Persona (70+ fields), CancellableTask, NavigationState

CHAPTER 4

Backend Architecture

Services, Routes, and Prompt Templates

Backend Architecture

ASGI Application Stack

The backend is a Quart application (async Flask) running under Hypercorn, an ASGI server. The ASGI stack layers python-socketio on top of the Quart app, enabling both HTTP and WebSocket communication through a single server process on port 5137.

A dedicated AI runner thread with its own asyncio event loop handles autonomous conversation execution, isolated from the main request-handling event loop. This avoids Motor (async MongoDB) event loop affinity issues and prevents long-running AI operations from blocking HTTP requests.

Service Layer

Business logic is organized into 19 service modules, each responsible for a specific domain. Services are stateless (except the AI Runner singleton) and communicate through function calls, the LLM service, and the WebSocket manager.

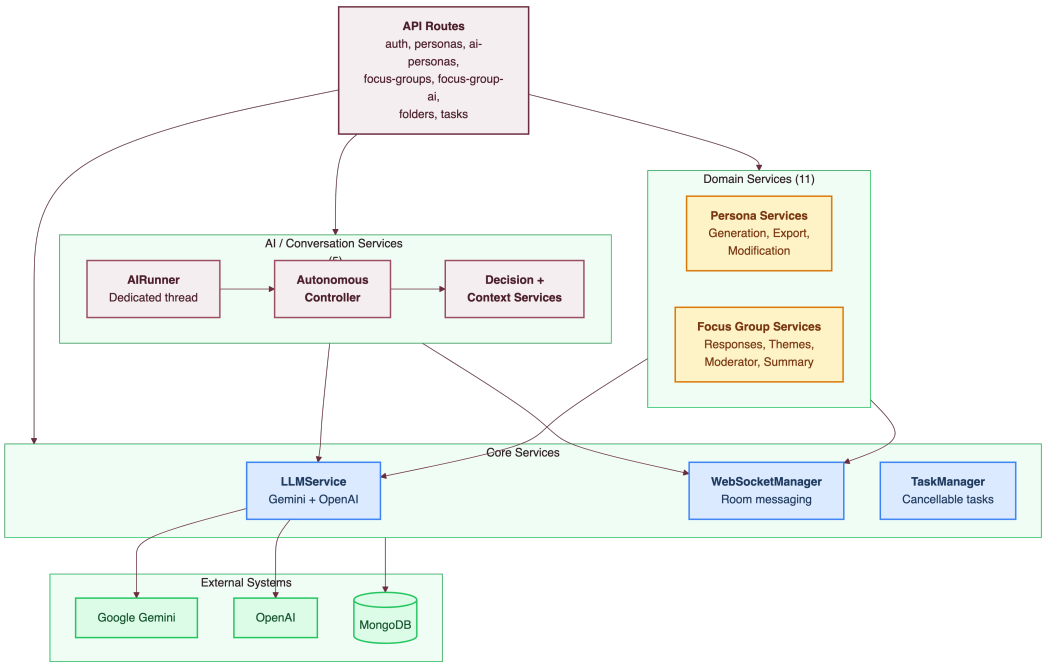

Figure 4.1 — Backend Service Architecture

Core Services

Service	File	Purpose
`LLMService`	llm_service.py	Multi-model abstraction (Gemini, GPT-4.1, GPT-5.2) with retry logic and JSON parsing
`WebSocketManager`	websocket_manager_async.py	Room-based messaging, event emission, connection tracking
`TaskManager`	task_manager.py	CancellableTask wrapper for long-running operations with per-user tracking
`PromptLoader`	utils/prompt_loader.py	Loads and interpolates 20 markdown prompt templates

AI / Conversation Services

Service	File	Purpose
`AIRunnerService`	ai_runner_service.py	Singleton: dedicated thread + event loop for autonomous conversations
`AutonomousConversation Controller`	autonomous_conversation_controller.py	State machine orchestrating multi-persona conversation flow
`ConversationDecision Service`	conversation_decision_service.py	LLM-driven decision engine: next speaker, action type, probing
`ConversationContext Service`	conversation_context_service.py	Aggregates messages, participants, and state for LLM context
`ConversationStateManager`	conversation_state_manager.py	Tracks participation metrics, sentiment, energy levels

Persona Services

Service	File	Purpose
`AIPersonaService`	ai_persona_service.py	Two-stage persona generation with customer data integration
`PersonaModification Service`	persona_modification_service.py	AI-assisted persona editing
`PersonaExportService`	persona_export_service.py	Individual persona profile export
`BulkExportService`	bulk_persona_export_service.py	Batch persona export (MD/JSON/CSV)
`CustomerDataService`	customer_data_service.py	Upload and integrate research data into generation

Focus Group Services

Service	File	Purpose
`FocusGroupService`	focus_group_service.py	CRUD operations and discussion guide generation
`FocusGroupResponse Service`	focus_group_response_service.py	Generate persona responses with personality-driven prompts
`KeyThemeService`	key_theme_service.py	AI-powered theme extraction from conversation messages
`AIModeratorService`	ai_moderator_service.py	AI moderator intervention and discussion guidance
`FocusGroupSummary Service`	focus_group_summary_service.py	Comprehensive session summary generation
`ImageDescription Service`	image_description_service.py	Multimodal image description for uploaded assets

Prompt Template System

The backend uses 20 markdown prompt templates stored in **/backend/prompts/**. The PromptLoader utility loads these files and interpolates context variables (persona data, conversation history, discussion guide) before sending them to the LLM service.

Template	Used By
persona-basic-generation.md	Stage 1 persona generation (demographics)
persona-detailed-generation.md	Stage 2 persona expansion (full profile)
persona-system.md	System prompt for persona-as-character responses
focus-group-response.md	In-session persona response generation
conversation-decision-engine.md	Autonomous mode: next action decision
conversation-participant-selection.md	Autonomous mode: speaker selection
ai-moderator-system.md	AI moderator system prompt
probe-generation-prompt.md	Probing question generation
key-theme-extraction.md	Theme extraction from conversation
discussion-guide-generation.md	Structured discussion guide creation
focus-group-summary-generation.md	Post-session summary generation
audience-brief-enhancement.md	Research brief AI enhancement
image-description.md	Multimodal image analysis
persona-interaction-prompt.md	Persona-to-persona interaction
persona-to-persona-response.md	Inter-persona conversation
persona-modification.md	AI-assisted persona editing
persona-summary-generation.md	Persona summary for display
persona-download-summary.md	Export summary format
persona-profile-export.md	Full profile export format
key-theme-system.md	Theme extraction system prompt

CHAPTER 5

5 Data Model

MongoDB Collections and Relationships

Data Model

Collections Overview

Semblance stores data in MongoDB across four primary collections. Documents use MongoDB ObjectId references for relationships rather than foreign key constraints.

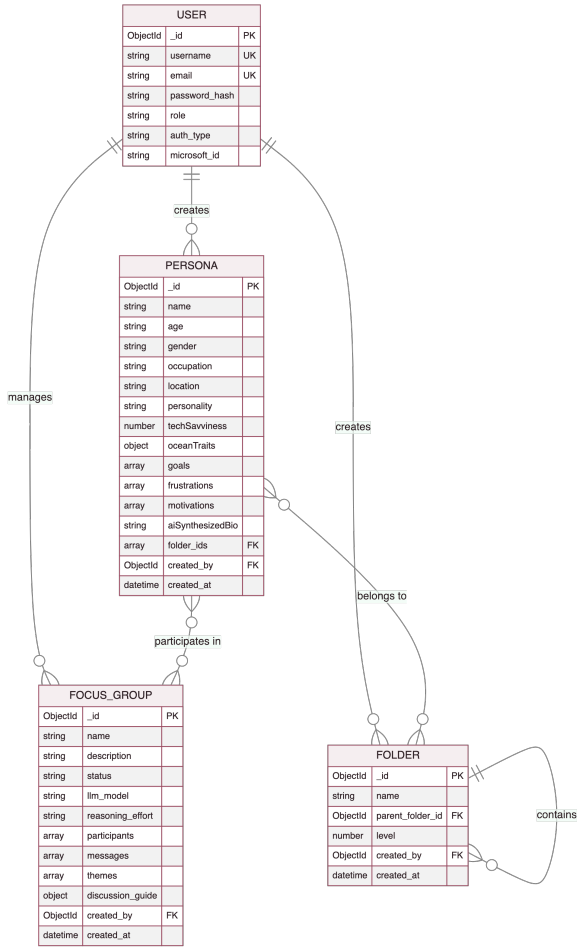

Figure 5.1 — Entity Relationship Diagram

User Collection

Stores authentication credentials and profile information. Supports dual auth types.

Field	Type	Description
_id	ObjectId	Primary key
username	String (unique)	Login identifier

Field	Type	Description
email	String (unique)	Email address
password_hash	String	bcrypt-hashed password
role	String	User role (default: "user")
auth_type	String	"local" or "microsoft"
microsoft_id	String	Azure AD object ID (optional)

Persona Collection

The richest data model with 70+ fields covering demographics, OCEAN personality traits, behavioral attributes, motivations, scenarios, and AI-generated content.

Core Fields

Field	Type	Description
_id	ObjectId	Primary key
name	String	Persona display name
age, gender, occupation	String	Demographics
location, education	String	Geographic and educational background
personality	String	Personality summary text
techSavviness	Number	Technology comfort level (0–100)
created_by	ObjectId	Reference to User who created this persona
folder_ids	Array[ObjectId]	Folders this persona belongs to (many-to-many)

Personality & Psychographic Fields

Field	Type	Description
oceanTraits	Object	OCEAN scores (0–100): openness, conscientiousness, extraversion, agreeableness, neuroticism
thinkFeelDo	Object	Arrays of thinks, feels, does statements
goals	Array[String]	Life and professional goals
frustrations	Array[String]	Pain points and frustrations
motivations	Array[String]	Driving motivations
selfDeterminationNeeds	Object	Autonomy, competence, relatedness assessments
scenarios	Array[String]	Behavioral scenario descriptions
aiSynthesizedBio	String	AI-generated narrative biography (2–3 lines)

Focus Group Collection

Stores session configuration, participant references, conversation messages, themes, and discussion guide. Messages and themes are embedded documents within the focus group.

Field	Type	Description
_id	ObjectId	Primary key
name, description	String	Session title and research topic
status	String	"new", "manual_mode", "ai_mode", or "completed"
llm_model	String	Selected LLM model (default: gemini-3-pro-preview)
reasoning_effort	String	GPT-5.2 reasoning level (minimal/low/medium/high)
participants	Array[Object]	Participant objects with persona_id references
messages	Array[Object]	Conversation messages (sender, text, timestamp, sentiment)
themes	Array[Object]	Extracted themes with supporting quotes
discussion_guide	Object	Structured guide with sections and items
created_by	ObjectId	Reference to User
autonomous_started_at	DateTime	Timestamp of autonomous mode start

Folder Collection

Hierarchical folder structure for organizing personas. Supports two-level nesting.

Field	Type	Description
_id	ObjectId	Primary key
name	String	Folder display name
parent_folder_id	ObjectId	Parent folder reference (null for root)
level	Number	Depth level (0 = root, 1 = child, max depth: 2)
created_by	ObjectId	Reference to User

Relationships

- **User** → **Persona** (1:N) — created_by field on Persona
- **User** → **Focus Group** (1:N) — created_by field on Focus Group
- **User** → **Folder** (1:N) — created_by field on Folder
- **Persona** ↔ **Focus Group** (M:N) — participants array in Focus Group references persona IDs
- **Persona** ↔ **Folder** (M:N) — folder_ids array on Persona references folder IDs
- **Folder** → **Folder** (1:N) — parent_folder_id for hierarchy

CHAPTER 6

6 Authentication

JWT, Microsoft OAuth, and WebSocket Auth

Authentication & Authorization

Dual Authentication System

Semblance supports two authentication methods that produce identical JWT tokens, allowing the rest of the application to be auth-method-agnostic:

- **Local JWT** — Username/password authentication with bcrypt password hashing. Tokens are HS256-signed with a 24-hour expiry.
- **Microsoft OAuth (MSAL)** — Azure AD authentication via MSAL popup flow. The frontend obtains an idToken from Azure, sends it to the backend for validation, and receives a Semblance JWT in return.

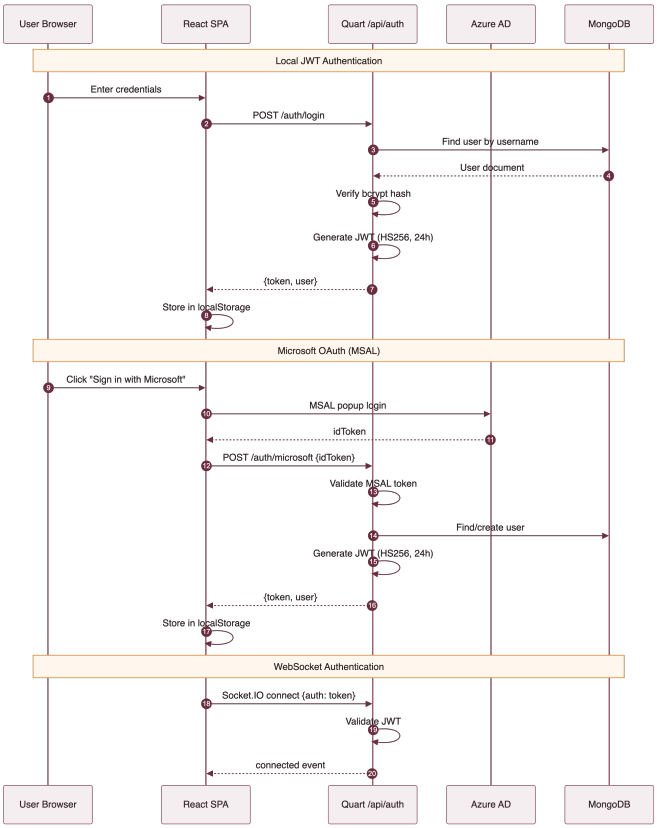

Figure 6.1 — Authentication Flow (Local JWT and Microsoft OAuth)

JWT Token Lifecycle

Property	Value
Algorithm	HS256
Expiration	24 hours
Claims	sub (user_id as string)
Storage	localStorage (auth_token key)
Header Format	Authorization: Bearer {token}
Validation	Checked on every API request via interceptor; validated server-side via @jwt_required decorator

Frontend Token Management

The Axios API client includes request and response interceptors for automatic token management:

- **Request interceptor** — Extracts JWT from localStorage, validates expiration by decoding the payload, attaches as Bearer token header.
- **Response interceptor** — Catches 401 responses, dispatches auth_error_event, clears localStorage, redirects to /login.
- **Session restoration** — On app mount, checks for existing token in localStorage, validates via GET /auth/me, restores session if valid.

WebSocket Authentication

Socket.IO connections authenticate via JWT token passed in the auth parameter during the connection handshake. The server validates the token before accepting the connection:

- Client sends: **auth: { token: jwt_token }** in Socket.IO connection options
- Server extracts and validates the JWT on the connect event
- On success: emits **connected** event with session info
- On failure: emits **auth_error** event and disconnects

Route Protection

Backend routes are protected with the **@jwt_required()** decorator (custom Quart-compatible implementation using PyJWT directly, not Flask-JWT-Extended). The decorator validates the Authorization header and makes the user identity available via **get_jwt_identity()**.

Frontend routes use the **ProtectedRoute** wrapper component that checks AuthContext isAuthenticated state and redirects unauthenticated users to /login.

CHAPTER 7

7

Real-Time Communication

WebSocket Architecture and Events

Real-Time Communication

Socket.IO Architecture

The application uses Socket.IO for bidirectional real-time communication between the React frontend and Quart backend. The backend uses python-socketio's AsyncServer (native ASGI compatibility), while the frontend uses the socket.io-client library wrapped in a singleton service.

Room-Based Messaging

Each focus group session has a dedicated Socket.IO room. When a user opens a session, the client emits a **join_focus_group** event, and the server adds the connection to the room. All subsequent events (messages, status updates, theme discoveries) are broadcast to the room, enabling multiple observers to watch a live session.

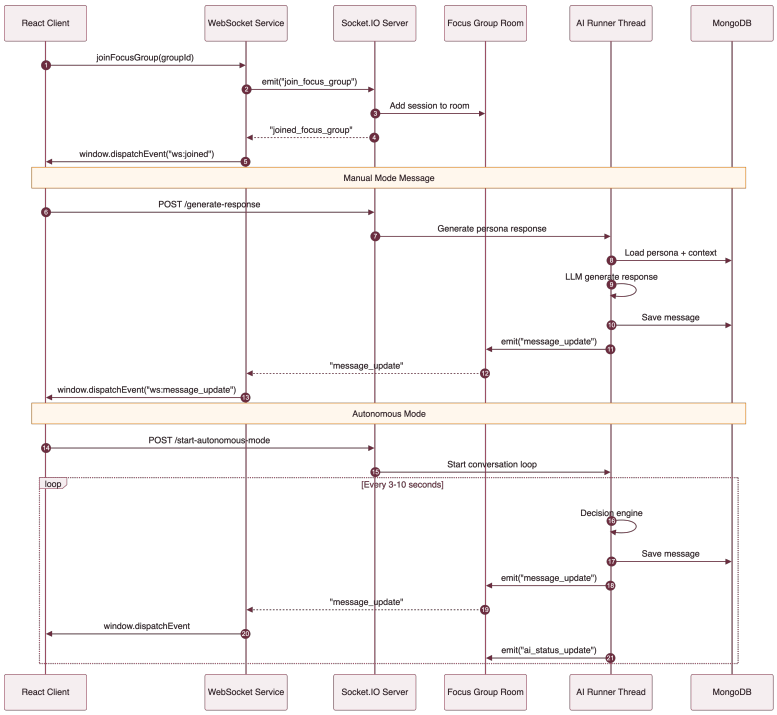

Figure 7.1 — WebSocket Communication Flow

Frontend Event Dispatching

The frontend uses a hybrid approach: the WebSocket singleton service binds specific listeners for known events, then re-dispatches all events as window CustomEvents with a **ws:** prefix. This decouples React components from the WebSocket implementation:

> **NOTE**
>
> Pattern: Socket.IO event "message_update" → window.dispatchEvent(new CustomEvent("ws:message_update", {detail: payload})). Components listen via window.addEventListener without needing direct socket references.

WebSocket Event Catalog

Client → Server

Event	Payload	Purpose
connect	auth: {token}	Authenticate WebSocket connection
join_focus_group	{focus_group_id}	Join session room
leave_focus_group	{focus_group_id}	Leave session room
cancel_task	{task_id}	Cancel running AI task

Server → Client

Event	Purpose
connected	Connection success confirmation with session info
auth_error	Authentication failure notification
joined_focus_group	Room join confirmation
message_update	New message in conversation (includes sender, text, timestamp)
ai_status_update	AI mode status change (running/paused/completed/error)
moderator_status_update	Moderator action notification
theme_update	Key theme discovered or updated
focus_group_update	Focus group properties changed
mode_event_update	Session mode switch (manual ↔ autonomous)
analytics_update	Conversation analytics data
conversation_state_update	Conversation state change
task_started / task_completed	Long-running task lifecycle events
task_cancelled / task_failed	Task termination events
bulk_export_progress	Export operation progress percentage

Reconnection Strategy

The Socket.IO client is configured with automatic reconnection. On reconnect, the service rebinds all event listeners (to survive reconnection cycles) and auto-rejoins the previous focus group room. Token refresh is attempted before reconnection to handle expired sessions.

Setting	Value
Transport	WebSocket only (no polling fallback)
Reconnection	Enabled (automatic)
Connection Timeout	60 seconds
Ping Interval	45 seconds
Ping Timeout	120 seconds

CHAPTER 8

8 AI/LLM Integration

Multi-Model Service and Task Management

AI/LLM Integration

Multi-Model LLM Service

The **LLMService** provides a unified interface for all AI operations, abstracting away provider differences between Google Gemini and OpenAI. Each call creates a fresh client instance to avoid event loop affinity issues in the async ASGI environment.

Model	Provider	API	Special Parameters
gemini-3-pro-preview	Google	genai.Client.generate_content()	temperature, max_tokens
gpt-4.1	OpenAI	chat.completions.create()	temperature, max_tokens
gpt-5.2	OpenAI	responses.create()	reasoning_effort (minimal/low/medium/high), verbosity (low/medium/high)

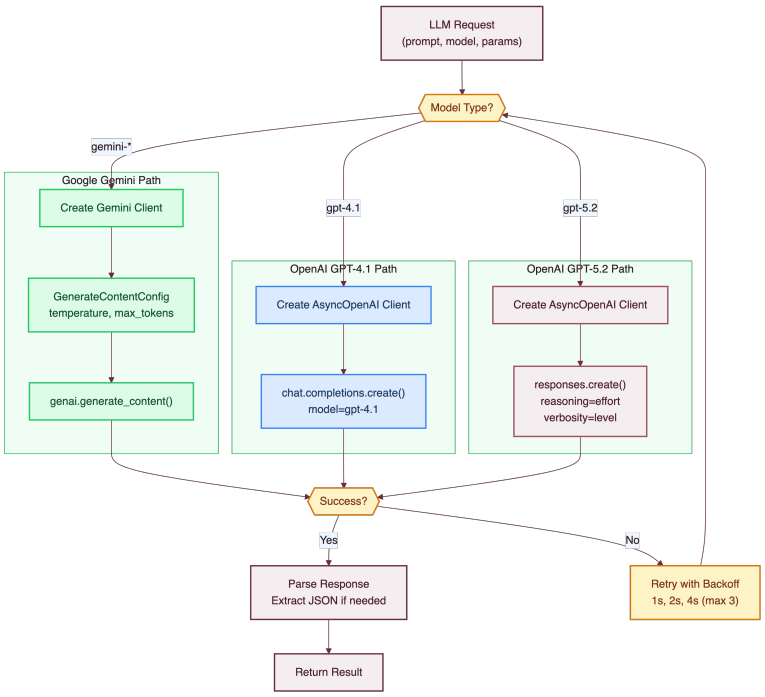

Figure 8.1 — LLM Request Pipeline

Retry and Error Handling

All LLM calls are wrapped in a retry mechanism with exponential backoff. On failure, the service retries up to 3 times with delays of 1s, 2s, and 4s. The service also handles JSON parsing of responses, stripping markdown code blocks when needed.

AI Runner Service

The **AIRunnerService** is a singleton that manages a dedicated background thread with its own asyncio event loop. This isolation ensures:

- Autonomous conversations don't block HTTP request handling
- Motor (async MongoDB driver) runs on a consistent event loop
- Thread-safe task registry enables concurrent conversation management

Lifecycle:

- **init_ai_runner()** — Called on app startup, creates dedicated thread and event loop
- **submit_conversation()** — Schedules conversation coroutine on the AI event loop
- **stop_conversation()** — Cancels a specific conversation by focus group ID
- **Shutdown** — Graceful cleanup: cancels all running tasks, stops event loop, joins thread

Task Management

Long-running operations (persona generation, theme extraction, summary generation) are wrapped in **CancellableTask** objects tracked by the TaskManager. Each task has a unique ID, is associated with a user, and can be cancelled via the REST API or WebSocket.

On the frontend, the **useCancellableGeneration** hook manages task lifecycle state and listens for WebSocket task events (task_started, task_completed, task_failed, task_cancelled).

CHAPTER 9

Core Feature Flows

Personas, Focus Groups, and Autonomous AI

Core Feature Flows

Persona Generation Pipeline

Persona generation uses a two-stage pipeline that progressively builds detail. The first stage generates basic demographic profiles from an audience brief, allowing the user to review before the second stage expands each into a fully detailed persona.

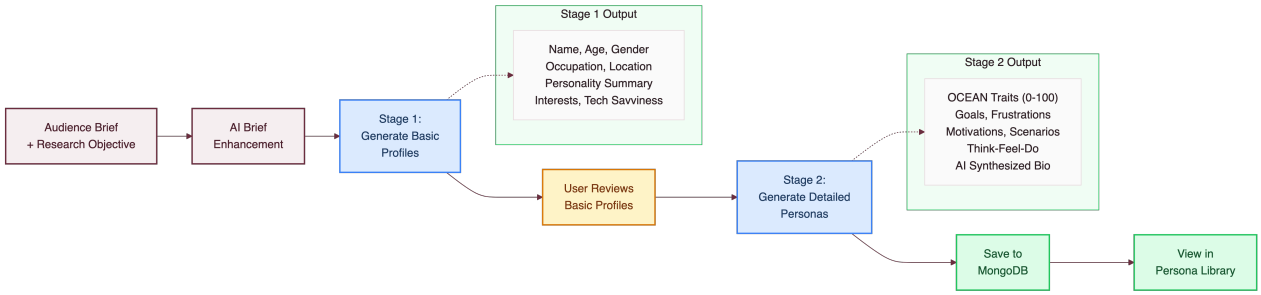

Figure 9.1 — Two-Stage Persona Generation Pipeline

Stage 1 — Basic Profiles: The LLM receives the audience brief (optionally AI-enhanced) and generates N basic profiles with name, age, gender, occupation, location, personality summary, interests, and tech savviness. Optional customer data files are included for grounding.

Stage 2 — Detailed Personas: Each basic profile is expanded individually. The LLM adds OCEAN personality traits (scored 0–100), goals, frustrations, motivations, behavioral scenarios, Think-Feel-Do mappings, self-determination needs, and an AI-synthesized biography. Each completed persona is saved to MongoDB immediately.

Focus Group Session Lifecycle

Focus groups progress through a state machine with four primary states. Users can switch between manual and autonomous modes during a live session.

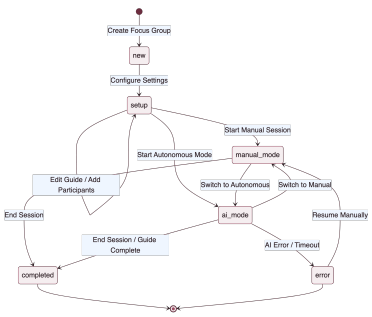

Figure 9.2 — Focus Group State Machine

State	Description	Transitions
new	Freshly created, no session started	Configure → setup
setup	Configuring guide, participants, settings	Start Manual or Start Autonomous
manual_mode	User-controlled moderation	Switch to AI, End Session
ai_mode	Autonomous AI-driven conversation	Switch to Manual, End Session, Guide Complete
completed	Session finished	Terminal state
error	AI error or timeout	Resume Manually, Terminal

Autonomous Conversation System

The autonomous conversation controller orchestrates multi-persona discussions without human intervention. It runs in a dedicated thread (via AIRunnerService) and uses the LLM-powered decision engine to determine each action.

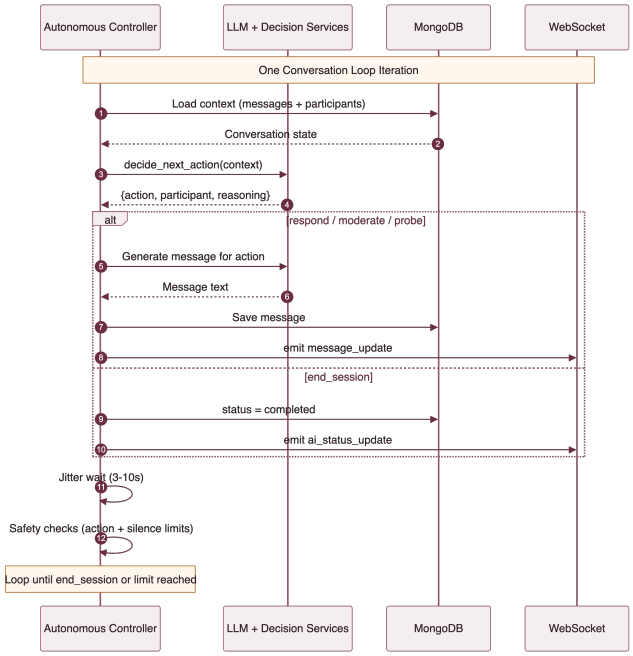

Figure 9.3 — Autonomous Conversation Loop (One Iteration)

Decision Engine Actions

Action	Description
participant_respond	Selected persona provides a contextual response
moderator_speak	AI moderator advances discussion or redirects
participant_interaction	Two personas engage in direct dialogue
probe_trigger	Probing question to deepen exploration
end_session	Conclude the conversation (guide complete or limits reached)

Safety Limits

Limit	Value	Purpose
Max Actions	500	Prevents runaway conversations
Max Consecutive Silence	3	Ends session if no meaningful responses
Response Timeout	30 seconds	Prevents hanging on unresponsive LLM
Inter-Action Delay	3–10 seconds (random)	Simulates natural conversation pace
Dominance Threshold	40%	Flags when a participant dominates discussion
Reasoning History	Last 20 decisions	Prevents repetitive decision patterns

CHAPTER 10

API Reference

REST Endpoint Catalog

API Reference

The backend exposes 7 route groups via Flask/Quart blueprints. All endpoints except authentication routes require a valid JWT token in the Authorization header.

/api/auth

Method	Path	Auth	Purpose
POST	/register	No	Register new user account
POST	/login	No	Login with username/password, returns JWT
POST	/microsoft	No	Microsoft MSAL authentication, returns JWT
GET	/me	Yes	Validate token and return user profile

/api/personas

Method	Path	Auth	Purpose
GET	/	Yes	Get current user's personas
GET	/all	Yes	Get all personas
GET	/:id	Yes	Get specific persona by ID
POST	/	Yes	Create new persona
POST	/batch	Yes	Create multiple personas
PUT	/:id	Yes	Update persona
DELETE	/:id	Yes	Delete persona
POST	/:id/export-profile	Yes	Export persona as formatted profile
POST	/bulk-export	Yes	Export multiple personas (MD/JSON/CSV)
POST	/:id/modify-with-ai	Yes	AI-assisted persona modification

/api/ai-personas

Method	Path	Auth	Purpose
POST	/generate-basic-profiles	Yes	Stage 1: Generate basic demographic profiles
POST	/generate-personas	Yes	Stage 2: Expand profiles to full personas
POST	/enhance-audience-brief	Yes	AI-enhance the audience brief
POST	/upload-customer-data	Yes	Upload research documents for grounding
DELETE	/cleanup-customer-data/:id	Yes	Clean up uploaded customer data
POST	/batch-generate-summaries	Yes	Generate summaries for multiple personas

/api/focus-groups

Method	Path	Auth	Purpose
GET	/	Yes	Get user's focus groups
GET	/:id	Yes	Get specific focus group
POST	/	Yes	Create focus group
PUT	/:id	Yes	Update focus group
DELETE	/:id	Yes	Delete focus group
POST	/:id/participants	Yes	Add participant to group
DELETE	/:id/participants/:pid	Yes	Remove participant
POST	/:id/messages	Yes	Send message in session
GET	/:id/messages	Yes	Get session messages
POST	/:id/assets	Yes	Upload creative assets
POST	/:id/generate-discussion-guide	Yes	Generate AI discussion guide
POST	/:id/notes	Yes	Create session note

/api/focus-group-ai

Method	Path	Auth	Purpose
POST	/generate-response	Yes	Generate persona response in session
POST	/generate-key-themes	Yes	Extract themes from conversation
POST	/autonomous/start/:id	Yes	Start autonomous conversation
POST	/autonomous/stop/:id	Yes	Stop autonomous conversation
GET	/autonomous/status/:id	Yes	Get autonomous mode status
POST	/moderator/advance/:id	Yes	Advance moderator to next topic
POST	/moderator/end-session/:id	Yes	End session via moderator
GET	/conversation/state/:id	Yes	Get conversation state
GET	/conversation/analytics/:id	Yes	Get conversation analytics
POST	/conversation/intervene/:id	Yes	Manual intervention in autonomous mode

/api/folders

Method	Path	Auth	Purpose
GET	/	Yes	Get folder hierarchy tree
GET	/:id	Yes	Get specific folder
POST	/	Yes	Create folder
PUT	/:id	Yes	Update folder
DELETE	/:id	Yes	Delete folder
POST	/:id/personas	Yes	Add persona to folder
DELETE	/:id/personas/:pid	Yes	Remove persona from folder
POST	/:id/personas/batch	Yes	Add multiple personas to folder

/api/tasks

Method	Path	Auth	Purpose
DELETE	/:task_id	Yes	Cancel a running task
GET	/user/:user_id	Yes	Get user's active tasks

CHAPTER 11

Data Flow

End-to-End Request and Conversation Patterns

Data Flow

End-to-End Request Flow

A typical user interaction follows this path through the system:

- **1. User Action** — Click, form submit, or navigation event in the React SPA
- **2. API Request** — Axios sends REST request with JWT Bearer token via the API client
- **3. Route Handler** — Quart blueprint validates JWT, extracts user identity, calls service layer
- **4. Service Processing** — Business logic executes: LLM calls, database operations, prompt templating
- **5. Data Persistence** — MongoDB operations via PyMongo (or Motor in AI thread)
- **6. WebSocket Broadcast** — For real-time operations, events are emitted to the focus group room
- **7. Response** — JSON response returned to frontend; WebSocket events dispatched as window events
- **8. UI Update** — React components re-render via TanStack Query cache invalidation or WebSocket event handlers

AI Conversation Data Flow

During an autonomous conversation, data flows through a specialized pipeline:

- **1. User starts autonomous mode** — POST /autonomous/start/:id
- **2. AI Runner** — Submits conversation coroutine to dedicated thread
- **3. Conversation Loop** — Controller runs continuously until end condition
- **4. Context Assembly** — ConversationContextService loads messages, participants, guide from MongoDB
- **5. Decision** — ConversationDecisionService sends context + prompt to LLM, gets structured action
- **6. Execution** — Controller executes action (generate response, moderate, probe)
- **7. Persistence** — Message saved to MongoDB focus group document
- **8. Broadcast** — WebSocketManager emits message_update and status events to room
- **9. Frontend** — Window events trigger React component updates in real time
- **10. Loop** — 3–10 second delay, then repeat from step 4

Key Architectural Patterns

Singleton WebSocket Service

The frontend creates a single Socket.IO instance at module level, shared across all components. This prevents multiple socket connections and ensures consistent event routing. Events are re-dispatched as window CustomEvents, decoupling React from the WebSocket implementation.

Dedicated AI Thread

Autonomous conversations run in a dedicated Python thread with its own asyncio event loop. This prevents long-running AI operations from blocking HTTP request handling and provides a stable event loop for Motor (async MongoDB driver).

Two-Pass Document Rendering

Focus group sessions use an event-driven architecture where database writes and WebSocket broadcasts happen atomically. The WebSocket manager maintains room membership, ensuring only connected observers receive updates.

Prompt Template Engine

All LLM prompts are externalized as markdown files in /backend/prompts/. The PromptLoader reads templates at runtime and interpolates context variables. This separation enables prompt iteration without code changes.

> **NOTE**
>
> This document was auto-generated from the Semblance codebase. For the most current details, consult the source code directly. Service files are in /backend/app/services/, route files in /backend/app/routes/, and frontend components in /src/components/.