**Project Report**

|  |  |
| --- | --- |
| **Qualification Name** | BDSE |
| **Module Name** | CAI |

**w**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student name** | | **Assessor name** | |
| Arcipe, Jhonn Vincent | |  | |
| **Date issued** | **Completion date** | | **Submitted on** |
| 1/17/26 |  | |  |
|  | | | |
| **Project title** | **Capstone Software Development Project** | | |

Date: 1/17/26

Student signature:

I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.

**Learner declaration**

**Index**

1. Project Overview

2. Background and Problem Statement

3. Project Proposal and Planning

4. System Design and Architecture

5. Data Preparation and Processing

6. Application Development and Testing

7. Visualization and Reporting (if applicable)

8. Deployment and Integration

9. Documentation and Presentation

10. References

11. Appendices

**Document Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Effective Date of release** | **Details** | **Author** |
| 1.0 | 15 Nov 2025 | Initial Creation |  |
| 1.1 | 16 Feb 2026 | Content revision aligned with current repository implementation and feature status | Jhonn Vincent Arcipe |

# Project Overview

### Introduction

In the contemporary software engineering landscape, there is a widening gap between the speed of code generation and the maturity of architecture design. While AI-assisted coding tools accelerate implementation, they often operate without complete system context. This creates "context loss," where high-level design intent drifts from actual code decisions over time.

Traditional diagram tools often produce static visual artifacts where components are shapes rather than semantically validated architecture units. As teams adopt distributed systems, keeping architecture, implementation, and AI-assisted coding outputs aligned becomes increasingly difficult without a structured, machine-readable blueprint.

Simulark is a generative architecture platform that converts natural-language requirements into semantic, interactive system diagrams. Unlike static diagram tools, each node and connection carries structural meaning that can be validated, simulated, exported, and reused by AI coding workflows. The platform combines semantic node modeling, protocol-aware flows, and machine-readable context outputs to reduce architecture-to-implementation drift.

## Project Objectives

**General Objective:** To design and develop a full-stack web application using Next.js 16 and Supabase that democratizes system architecture design through Generative AI.

### Specific Objectives:

1. To implement a multi-provider architecture generation engine that streams responses, validates output, and converts prompts into usable graph JSON.
2. To develop an interactive XYFlow canvas with semantic node types, auto-layout, and high-performance graph editing for real-world architecture design.
3. To enforce architecture correctness with Valibot schemas and post-generation validation rules (including auto-fix support for common graph issues).
4. To implement tier-aware usage controls and project persistence using Supabase, including daily generation limits and project version snapshots.

## Scope of the Project

This study focuses on the development of **Simulark**, a "Generative UI" platform designed to bridge the gap between architectural design and code generation by transforming natural language prompts into active, semantic architecture diagrams.

The project centers on creating a "Living Blueprint" that moves beyond static shapes to provide developers with machine-readable context for AI coding agents. The system will be beneficial for the following stakeholders:

* + **Target Users:** The platform is designed to serve a diverse range of technical roles, including **Software Architects** who require high-level design visualization, **Tech Leads** managing system complexity, and **Junior Developers** who need a "Living Blueprint" to understand distributed systems.
  + **Platform:** Simulark is developed as a **desktop-first web application** optimized for modern browsers such as **Chrome**, **Edge**, and **Firefox**, ensuring high performance for its interactive canvas and WebGL rendering requirements.
  + AI Integration: The platform routes requests across NVIDIA, ZhipuAI, OpenRouter, and Kimi-compatible endpoints depending on selected model and fallback rules, while maintaining a consistent streaming UX.

### Limitations of the Study

While Simulark provides a robust framework for architectural visualization, the following limitations define the current boundaries of the study:

* + **Infrastructure Provisioning:**The system is strictly limited to generating architectural designs and semantic context. It does not currently support the automated provisioning of real-world cloud resources or the generation of Infrastructure-as-Code (IaC) files, such as Terraform or AWS CDK.
  + Internet Connectivity: Core generation requires active connectivity to external AI providers (NVIDIA, ZhipuAI, OpenRouter, and related model endpoints). Cached or saved diagrams can still be viewed and edited offline.

### Methodology

The development of **Simulark** was executed using an **Agile Software Development Methodology**, specifically adopting an **iterative and incremental approach**. This framework was chosen to effectively manage the complexities involved in synchronizing non-deterministic **AI orchestration** with a high-performance, **real-time web interface**, allowing for continuous refinement of the generative engine and canvas interactions.

The development lifecycle was divided into three distinct sprints:

* + **Sprint 1:** The primary focus was establishing the technical bedrock of the application. Key deliverables included the configuration of **Next.js 16** and the integration of **Supabase** for secure authentication using **Row-Level Security (RLS)**. The foundational interactive layer was completed by implementing the **xyflow** engine to support basic node drag-and-drop functionality and canvas persistence.
  + Sprint 2: This phase focused on production AI runtime integration. Core generation was implemented through Next.js API routes (/api/generate and /api/chat) with streaming responses, prompt validation, model routing, and structured JSON recovery when outputs are noisy. A thinking/streaming UI was added to improve transparency during long-running generation.
  + Sprint 3: This phase prioritized simulation depth and reliability. The team refined protocol-aware edge behavior, chaos mode interactions, auto-layout strategies, and export flows. Persistence and versioning were hardened through Supabase project storage and snapshot logic in save operations.

# Background and Problem Statement

### Context and motivation

The motivation for this project comes from the growing complexity of distributed software systems. As applications evolve from monolithic designs to service-based architectures, maintaining a shared mental model across the team becomes difficult, and design/implementation drift becomes common.

### Problem description

The study aims to address the critical issue of "Context Loss" in the modern software development lifecycle, where high-level architectural design often fails to keep pace with rapid code generation. Specifically, it seeks to answer the following:

1. **Architectural Divergence (Context Loss):** While AI coding assistants accelerate development, they often lack a "big picture" understanding, leading to code that is disconnected from the intended system architecture.
2. **Static Visualization Limitations:** Existing design tools (e.g., Lucidchart or Visio) produce static shapes without semantic meaning, making it impossible to simulate data flow or validate connections automatically.
3. Translation Complexity: Architects and developers must manually convert natural-language requirements into technical architecture structures, which is error-prone and slow without assistive tooling.
4. Non-Deterministic AI Output: Generative models can produce incompatible or incomplete architecture structures unless constrained by schema validation, post-processing, and recovery logic.

### Assumptions

The development and evaluation of Simulark are based on the following technical and operational assumptions:

* + **Single-Tenant Focus:** It is assumed that the initial deployment will prioritize individual project workflows. While the system architecture is designed for scalability, real-time multi-user collaboration (multiplayer editing) is considered outside the immediate objectives of the current development cycle.
  + **Browser Capabilities:** To ensure the successful rendering of the high-performance interactive canvas, it is assumed that users are utilizing modern web browsers with **WebGL** and hardware acceleration enabled.
  + Persistent Connectivity: Generation workflows depend on external AI provider APIs; therefore, stable internet access is required for full functionality. Saved projects remain viewable/editable even when generation endpoints are unavailable.

# Project Proposal and Planning

Timeline

|  |  |  |
| --- | --- | --- |
| **Phase** | **Dates** | **Deliverables** |
| **Week 1: The Core** | **Jan 3 – Jan 9** | Setup and Core Architecture | Initialize Next.js 16 + Bun runtime + Supabase integration, secure auth with RLS, and baseline XYFlow editing interactions. | Goal: persistent architecture workspace with authenticated user projects. |
| **Week 2: The Brain** | **Jan 10 – Jan 16** | AI Runtime Integration | Implement streaming generation routes (/api/generate, /api/chat), prompt validation, model routing, and fallback behavior across providers. | Goal: reliable text-to-graph generation with real-time assistant feedback. |
| **Week 3: The Polish** | **Jan 17 – Jan 22** | Simulation, Export, and Reliability | Add protocol-aware flows, chaos interactions, auto-layout, context bridge exports, and skill packaging. Harden persistence with project snapshots/versioning. | Goal: production-capable end-to-end architecture workflow. |
| **Final Prep** | **Jan 23 – Jan 24** | Documentation, QA, and Defense | Consolidate technical report, validate core test suites, and prepare final demonstration narrative. | Goal: final submission with implementation-aligned evidence. |

**Stakeholders**

The success of **Simulark** relies on addressing the needs of various technical and organizational groups. These stakeholders are categorized into primary, secondary and tertiary roles:

**1. Primary Stakeholders (Direct Users)**

* **Software Architects**: These are the lead users who utilize Simulark to bridge the gap between abstract system requirements and structured visual design. They rely on the tool to generate "Living Blueprints" that serve as a single source of truth for the team.
* **Technical Leads and Senior Developers**: They use the platform to quickly prototype complex distributed systems and ensure that the AI-generated architecture follows best practices before moving into the coding phase.
* **Junior Developers**: This group benefits from the platform’s ability to provide high-level context. By visualizing how different services (Gateways, Queues, Databases) interact, they can overcome the "Context Loss" often associated with AI-driven development.

**2. Secondary Stakeholders (System & Business Impact)**

* **Project Managers / Scrum Masters**: While not direct designers, they benefit from the clarity the tool provides during sprint planning. A clear architectural diagram reduces technical debt and helps in estimating the complexity of tasks.
* **Quality Assurance (QA) Engineers**: They use the semantic diagrams to understand the flow of data across the system, allowing them to design more accurate integration tests based on the "traffic simulation" generated by the platform.
* AI Service Providers (NVIDIA, ZhipuAI, OpenRouter, Kimi-compatible endpoints): These providers are technical dependencies whose availability, rate limits, and model behavior directly affect generation quality and response latency.

**3. Tertiary Stakeholders**

* Academic Evaluators: Faculty and thesis advisors assess the technical methodology, implementation rigor, and how effectively the platform aligns system design outputs with practical engineering workflows.

**Tools and Technology**

|  |  |  |
| --- | --- | --- |
| **Category** | **Technology** | **Purpose** |
| **Frontend** | Next.js 16, React 19, Tailwind CSS v4, XYFlow, Zustand | Interactive architecture editor, responsive dashboard UI, and client-side simulation state |
| **Backend** | Bun runtime, Next.js API routes/Server Actions, Supabase PostgreSQL + Auth | Streaming generation endpoints, project/chat persistence, and secure multi-tenant data access |
| **AI** | OpenAI SDK + provider integrations (NVIDIA, Zhipu, OpenRouter, Kimi-compatible) | Model routing, fallback execution, and architecture generation pipelines |
| **Tools** | Biome, Vitest, TypeScript strict mode, Valibot | Code quality, schema validation, and automated test coverage for core modules |

## Task Coverage Matrix (Activities 1-8)

The matrix below maps the template Activity 1-8 requirements to implemented Simulark deliverables and report evidence.

|  |  |  |
| --- | --- | --- |
| Template Activity | Simulark Implementation Coverage | Report Evidence |
| 1. Proposal and Planning | Timeline, stakeholder analysis, scope, and technical stack were defined before implementation. | Section 3, timeline table, stakeholder discussion |
| 2. System Design and Architecture | Semantic graph architecture, API layer, AI runtime, and database model were designed and documented. | Section 4, architecture narratives, ERD and component breakdown |
| 3. Data Preparation and Processing | Prompt/context preprocessing, model routing, JSON recovery, and graph validation pipeline implemented. | Section 5, data processing and generation pipeline details |
| 4. Application Development | Frontend canvas, assistant panel, project APIs, and persistence workflows implemented end-to-end. | Section 6, frontend/backend/database implementation evidence |
| 5. Testing and Validation | Unit and schema tests executed across AI client, resilience, circuit breaker, prompt logic, and API schemas. | Section 6 testing methodology and test evidence tables |
| 6. Visualization and Reporting | Protocol-aware flows, context bridge exports, Mermaid output, and report-ready artifacts implemented. | Section 7 (Visualization and Reporting), export and context features |
| 7. Deployment and Integration | Vercel deployment flow, Supabase integration, security headers, and runtime health checks configured. | Section 8, deployment strategy and integration evidence |
| 8. Documentation and Presentation | Comprehensive capstone report prepared with outcomes, lessons, future work, references, and appendices. | Section 9 onward, references and appendices |

# System Design and Architecture

Simulark is an AI-powered Generative UI platform designed to bridge the gap between high-level system design and low-level implementation. The platform addresses the “Context Loss” problem inherent in modern software engineering, where architectural intent is often lost when transitioning to AI-assisted coding. By transforming natural language requirements into semantic, auto-arranged diagrams with active visual data flows, Simulark acts as a high-fidelity Computer-Aided Design tool specifically engineered for backend development.

**Frontend Architecture**

The frontend is built on Next.js 16 using the App Router for server-side rendering and efficient routing. TypeScript provides strict type safety across the entire application, while Tailwind CSS v4 offers a high-performance, utility-first design system. The core of the visualization engine relies on XYFlow (formerly React Flow), which has been heavily customized to provide an interactive architecture canvas. Unlike generic shape tools, Simulark uses specialized nodes with strict semantic definitions for gateways, compute services, databases, and message queues. This ensures the underlying data model remains structurally valid and prevents invalid architecture configurations from being represented. Zustand manages the global client-side state, handling the complex interactions of the simulation engine, including node positioning, edge connections, and real-time visual updates.

**AI Orchestration System**

The AI pipeline is implemented with production route handlers and client abstractions rather than a separate aggregator file. Requests are validated, routed by selected model/provider, streamed incrementally, and then parsed into architecture JSON. The runtime includes retry logic, circuit-breaker protections, and validation passes to improve reliability under unstable provider conditions.

**Multi-Provider AI Strategy**

Simulark applies a provider-aware strategy across NVIDIA-hosted models, Zhipu, OpenRouter, and Kimi-compatible APIs. Default routing favors performant baseline models, with fallback behavior for rate limits or provider errors. This design supports continuity while keeping model selection flexible by tier and user preference.

**Graph Data Model and Validation**

The architecture is represented as a directed graph using Valibot for runtime schema validation. The graph consists of nodes that represent system components such as gateways, services, databases, queues, and AI components, along with edges that represent the connections and data flows between them. Each node type carries specific metadata including labels, technology choices, descriptions, validation status, and cost estimates. Edges are protocol-aware, distinguishing between synchronous operations like HTTP and gRPC, and asynchronous patterns like message queues and event streams. This semantic approach ensures that the diagrams are not merely visual representations but serve as accurate technical documentation that can be validated against architectural best practices.

**Backend Services and API Design**

The backend exposes endpoints for generation, chat, project CRUD, skill export, health checks, and context retrieval. Streaming endpoints power the assistant UX, while context features are available through live context URLs and copyable exports such as .cursorrules and Mermaid. Current implementation includes both production context endpoints and one legacy mock-style context route retained for compatibility.

**Database and Authentication Layer**

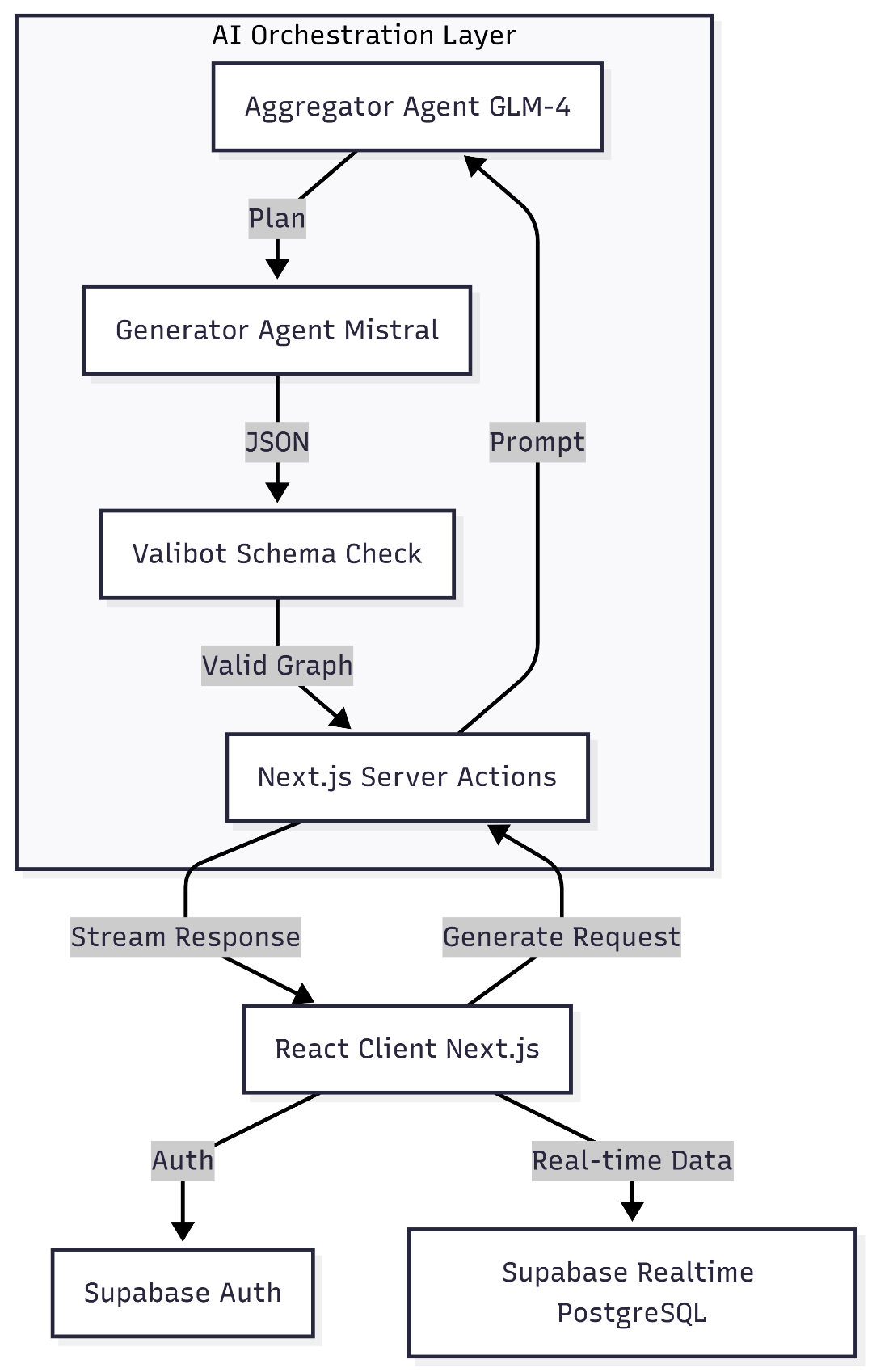
Supabase provides authentication, persistence, and relational storage with RLS. Core tables include users, projects, project\_versions, chats/chat\_messages, and usage tracking. Daily generation limits are enforced in the server layer based on plan (Doodle: 30/day, Sketch: 50/day, Blueprint: 1000/day), and usage is updated during generation requests.

**Interactive Features and Simulation**

The platform moves beyond static diagrams by visualizing the runtime behavior of systems. Data flows are animated to strictly represent their protocol nature, with synchronous operations like HTTP and gRPC looking visually distinct from asynchronous patterns like queues and streams. This allows architects and developers to instantly recognize blocking versus non-blocking paths in their architecture. The chaos mode feature provides a gamified simulation environment where users can test system fault tolerance by interacting with "kill switches" on any node, triggering simulations that demonstrate how traffic reroutes or where bottlenecks emerge when a component fails.

**Project Archetypes and Smart Tech Selection**

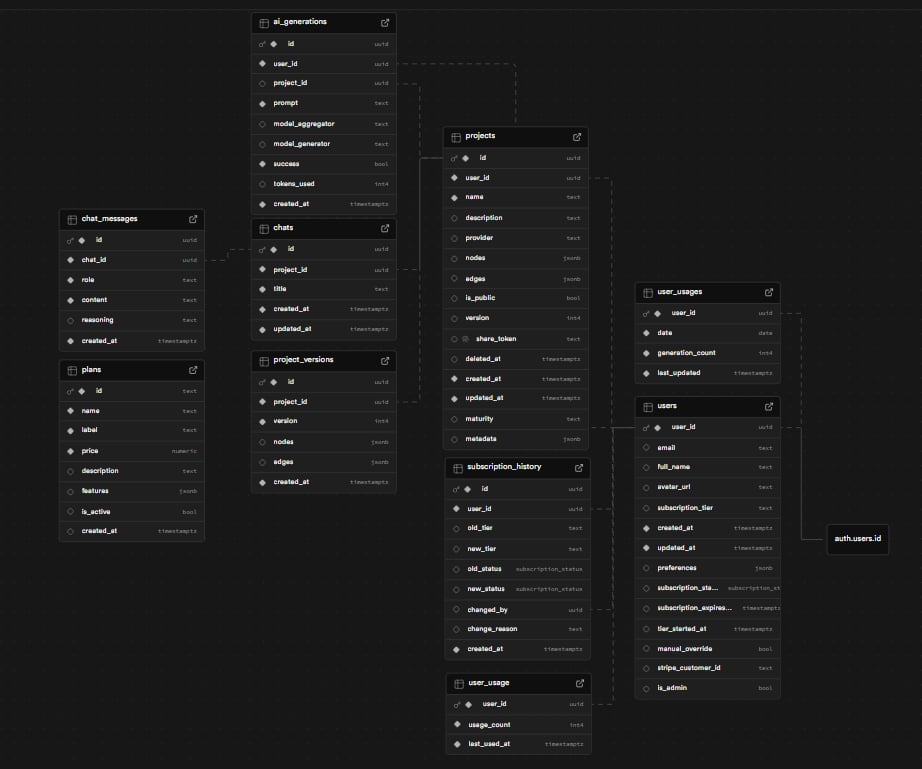
The generation system supports three operating modes: startup, default, and enterprise. Startup emphasizes managed simplicity and speed; default balances practicality and modern best practices; enterprise favors resilience, observability, and governance patterns. These modes influence prompting constraints and resulting architecture recommendations.



## Database Design (Data Model)

The production implementation uses Supabase (PostgreSQL) as the primary system of record for users, projects, chats, usage limits, and subscription lifecycle data. The schema and models below are aligned with current migrations and application validators.

#### Entity Relationship Diagram (ERD)



3.2 Implementation-Aligned Schema and Models

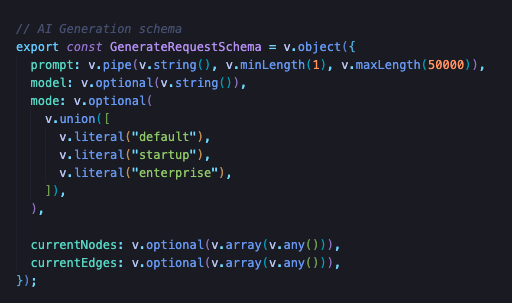
The following schema/model summary is verified against active Supabase migrations and TypeScript validators in this repository.

* + Architecture data model scope includes identity, project graphs, version snapshots, generation telemetry, chat persistence, subscription lifecycle, and rate-limit counters.

Important note: there is no separate subscription\_tiers table in the current schema. Tier logic is represented through users.subscription\_tier plus application plan configuration.

* + - users (PK: user\_id): email, full\_name, avatar\_url, subscription\_tier, subscription\_status, subscription\_expires\_at, tier\_started\_at, manual\_override, stripe\_customer\_id, is\_admin, onboarding\_completed, onboarding\_step, onboarding\_skipped, onboarding\_data, preferences, created\_at, updated\_at.
    - projects (PK: id, FK: user\_id): name, description, provider, nodes, edges, metadata, is\_public, version, share\_token, deleted\_at, created\_at, updated\_at.
    - project\_versions (PK: id, FK: project\_id): immutable version snapshots of nodes and edges for rollback and diffing.
  + ai\_generations (PK: id, FK: user\_id/project\_id): prompt, model\_aggregator, model\_generator, success, tokens\_used, created\_at for observability and diagnostics.
    - chats and chat\_messages: project-scoped conversations with role/content/reasoning storage and cascade delete behavior.
    - subscription\_history: auditable tier/status changes (old/new values, actor, reason, timestamp).
    - usage tracking tables:
    - user\_usages: active app-level daily counters (generation\_count/date/last\_updated), used by server checkRateLimit() flow.
    - user\_usage: legacy SQL/RPC counter path retained for compatibility (check\_and\_increment\_usage function).

Code Snippet A: GenerateRequestSchema (lib/schema/api.ts)



Code Snippet B: NodeTypeSchema Excerpt (lib/schema/graph.ts)



Code Snippet C: Connectors

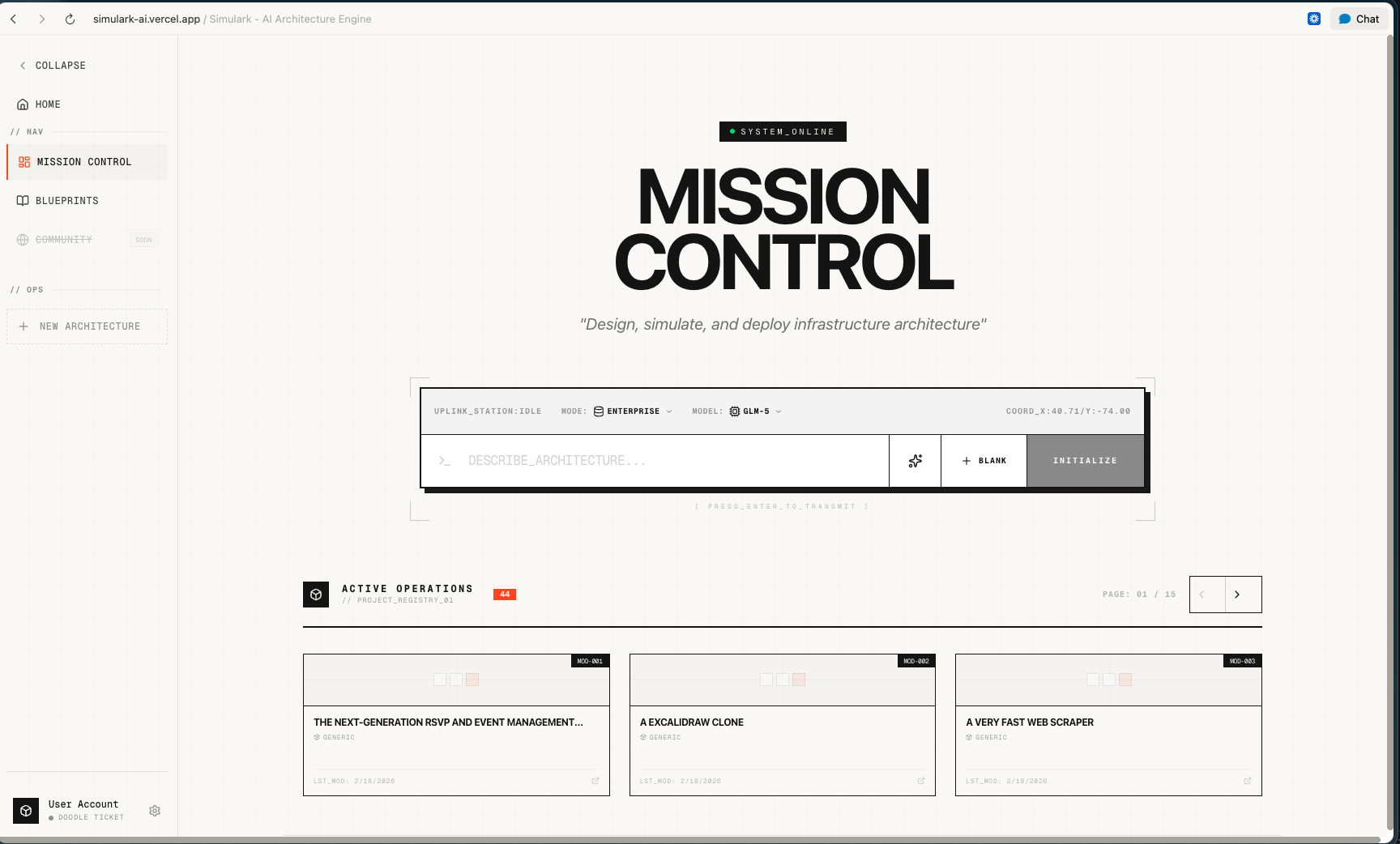


## User Interface Design (Frontend)

**4.1 Mission Control Dashboard Prototype**

The core screen provides immediate situational awareness for ongoing design and simulation tasks.

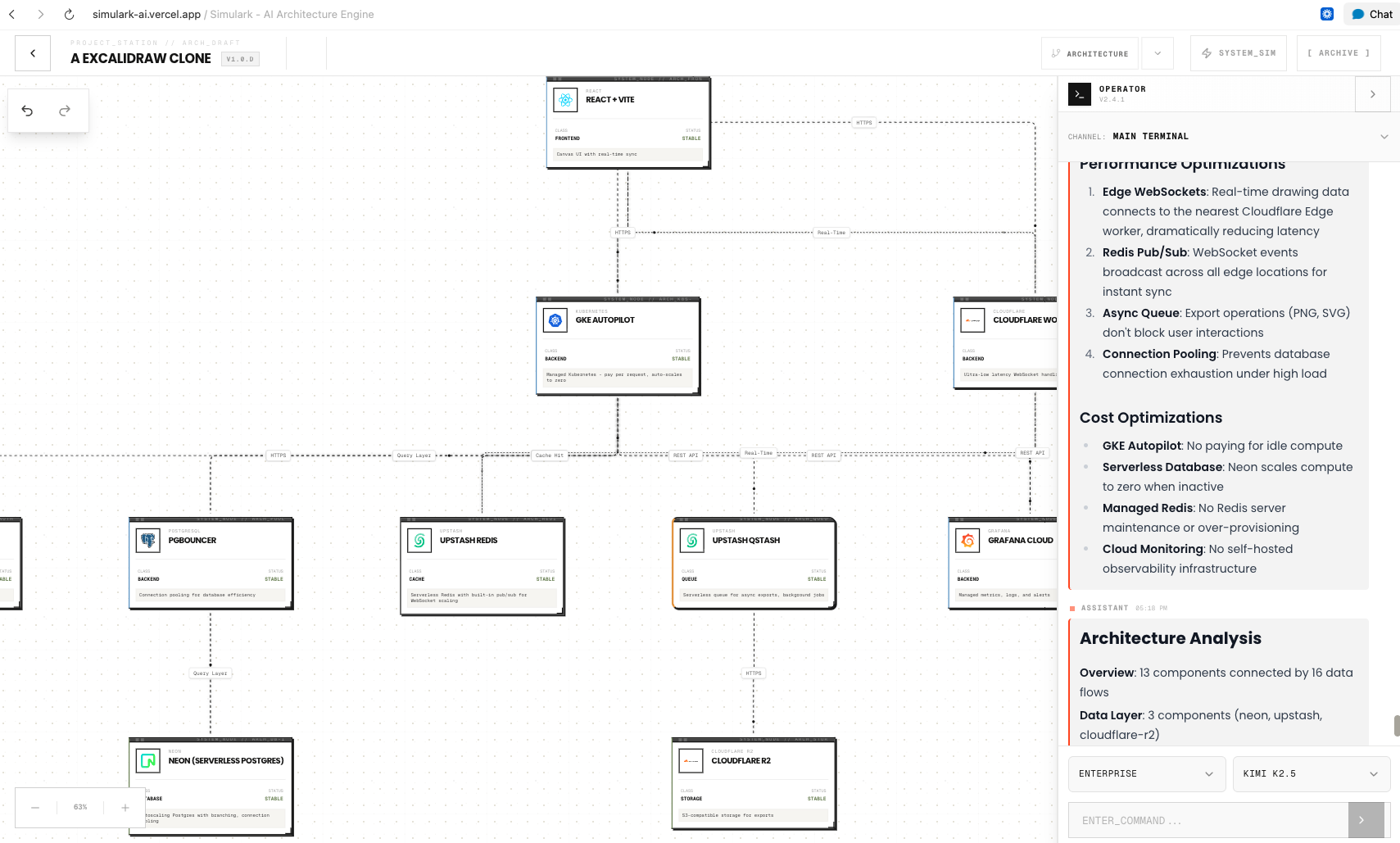
* **System Navigation:** A persistent left-hand sidebar that categorizes the workflow into "System Navigation" (Mission Control, Blueprints) and "Operations," allowing users to pivot between high-level overviews and specific builds.
* **Command Bar:** A central, prominent text input field labeled "Describe architecture..." that serves as the primary entry point for AI-driven generation, featuring quick-action buttons like "Blank" and "Run."
* **Active Operations Gallery:** A grid of cards below the main input that displays ongoing or recent projects (e.g., Chess diagram, E-commerce diagram). This allows users to monitor multiple live "operations" at a glance.
* **Visual Hierarchy:** The interface uses a top-down focus; the system status ("System Ready") and main title occupy the top center, while active tasks are organized into modular cards, ensuring the most important creative tools are always front-and-center.



**4.2 Workspace Interface Prototype**

The core screen (shown in the image) provides a high-fidelity environment for detailed architectural generation and manual editing.

* **Design Canvas**: A dominant, centralized workspace featuring a light grid pattern and zoom controls at the bottom-left, allowing for the precise placement of architectural components.
* **Creative Toolbar**: A vertical toolbar on the far left that provides essential design tools, including selection, hand panning, framing, shapes, text, and image imports.
* **System Terminal**: A dedicated right-hand sidebar that manages the AI interaction, featuring a "Select Terminal" dropdown and a placeholder stating, "Describe your architectural requirements to begin generation".
* **Interactive Command Input**: A bottom-right input field where users can enter requirements, select specific AI models (e.g., GLM-4.7-FLASH), and attach files to guide the generation process.
* **Global Navigation & Deployment**: A top header bar that displays the project status ("Untitled Operations"), file menus, system latency (12ms), and a prominent "Deploy" button for finalizing architectures.



#### AI Processing (Backend)

1: User Input

* User provides a natural language prompt (e.g., "Build a chat app with React and Python")
* System validates input and checks rate limits based on subscription tier

2: Aggregator Thinking (Step 1 of 2)

* **Model**: GLM 4.7
* LLM analyzes the request and creates a high-level architectural plan
* Identifies components: Gateway, Service, Database, Queue, etc.

3: Generator Processing (Step 2 of 2)

* **Models**: mistralai/mistral-small-3.1-24b-instruct:free or google/gemma-3-27b-it:free
* Converts the architectural plan into structured JSON
* Output includes nodes (components) and edges (connections) with protocols

4: Validation & Enrichment

* Validates JSON against **[ArchitectureGraphSchema](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/schema/graph.ts)**
* Enriches nodes with tech ecosystem data (icons, normalized tech IDs)
* Fallback safe architecture if generation fails

5: Response Delivery

* Streams response to frontend via SSE
* Separates reasoning (LLM thinking) from content (JSON)
* Frontend renders interactive architecture diagram

# Data Preparation and Processing

### Data Sources

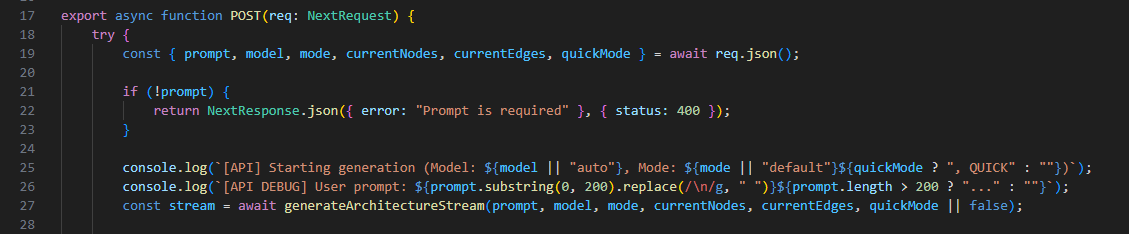
Simulark uses operational data rather than a traditional static dataset. Core inputs include user prompts, existing graph context (nodes/edges), model selection, and mode preferences. System-level metadata from Supabase and normalized technology mappings from the internal ecosystem catalog are used to enrich and validate generated outputs.

Processing Pipeline:

Input validation: Prompt quality is validated before generation requests are accepted.

Context assembly: Current architecture state (if any) is injected so modifications remain context-aware.

* **Model routing:** Provider/model resolution routes requests to NVIDIA, Zhipu, OpenRouter, or compatible endpoints.
* **Streaming generation:** The assistant streams reasoning/content chunks to the client for transparent progress.
* **JSON recovery:** Partial or noisy model outputs are parsed and recovered into valid graph structures where possible.
* **Architecture validation:** Generated graphs are validated against schema and architecture rules with auto-fix support.
* **Normalization and persistence:** Node metadata is enriched (icons/tech IDs) and then persisted to project storage.



Provider and Fallback Strategy

Primary routing paths:

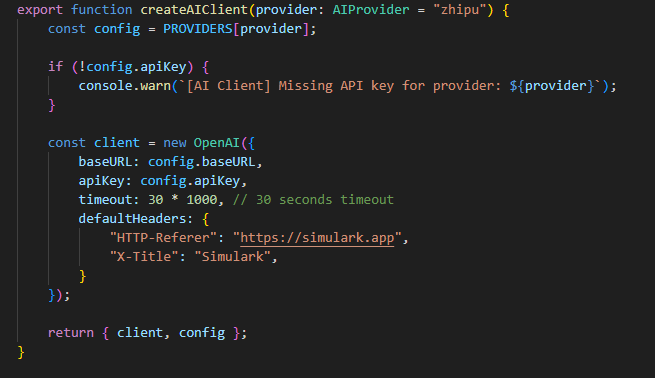
* nvidia: Default route for GLM-5 and selected advanced models (including NVIDIA-routed MiniMax/Kimi IDs).
* zhipu/openrouter: Direct provider routes and fallback paths when model/provider errors or rate limits occur.



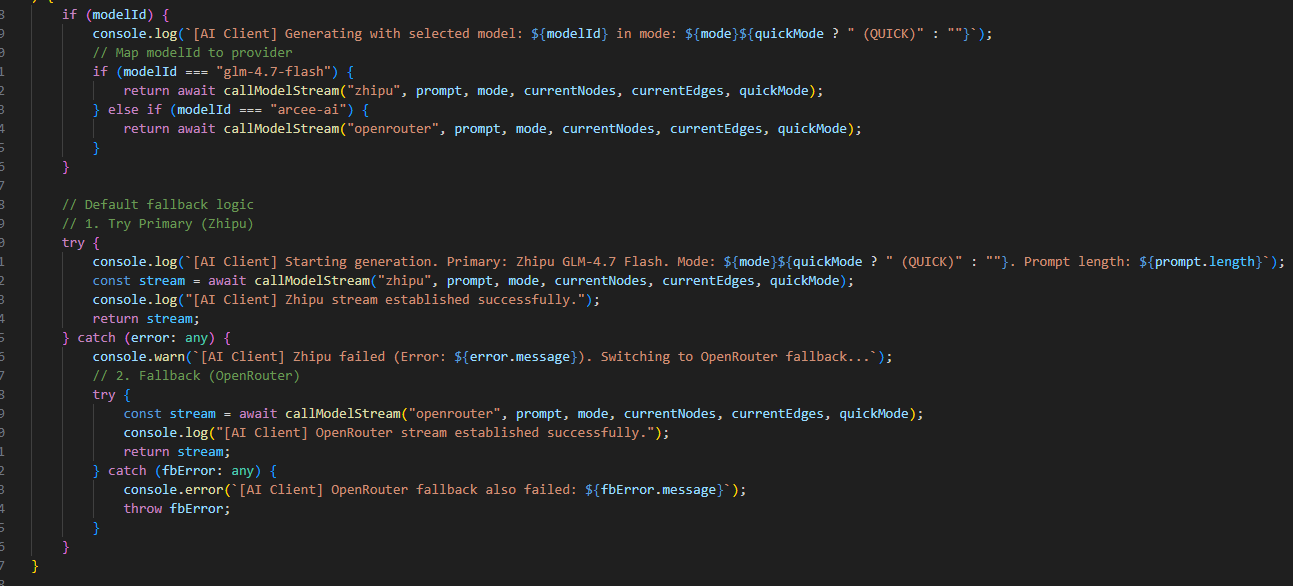
Provider Configuration (Environment-Driven)



Client Initialization and Timeout Controls



Fallback and Recovery Logic



Architecture Graph Output Contract

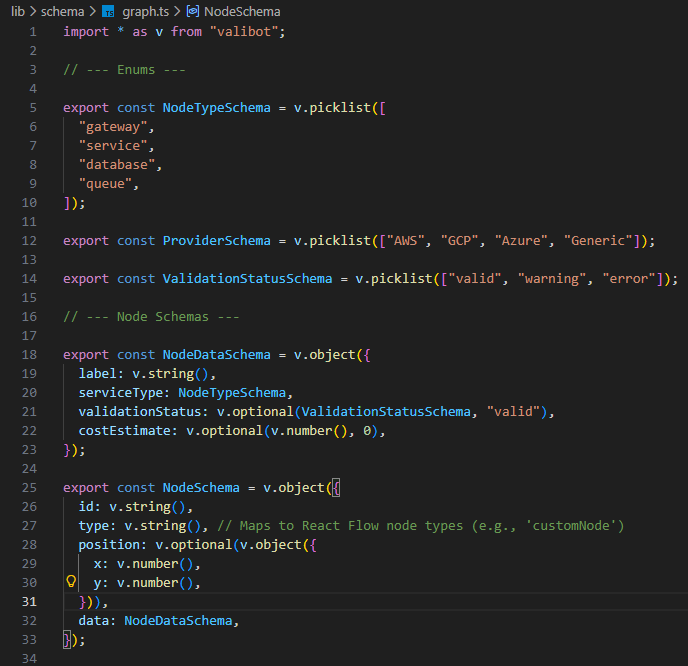
Node object requirements:

* id – Unique identifier
* type - semantic architecture type (for example: gateway, service, database, queue, auth, payment, monitoring, ai-model, vector-db)
* position – { x, y } coordinates
* data.label – Display name
* data.tech – Tech stack (nextjs, postgres, aws, etc)
* data.description – Component purpose

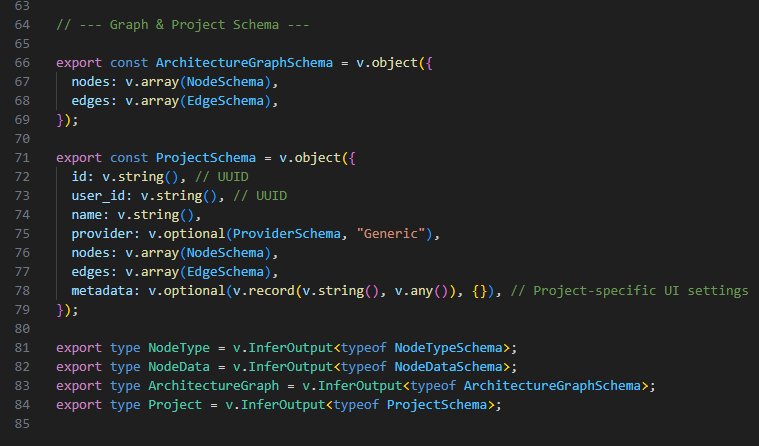
**Edges:**

* id – Unique identifier
* source/target – Node IDs
* data.protocol - transport or interaction protocol (for example: http, https, grpc, websocket, queue, stream, database, cache, oauth)

Schema Definitions



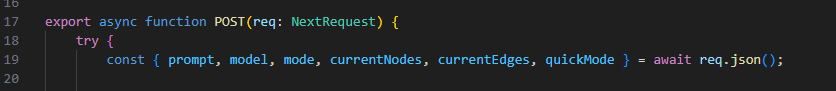




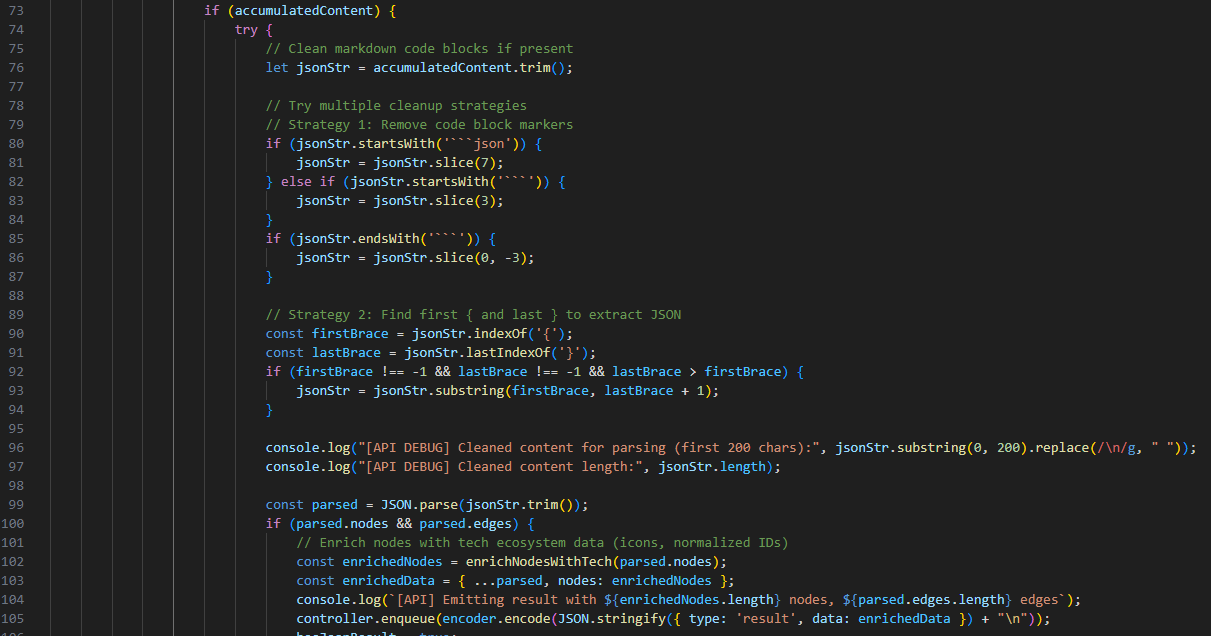
AI Prompt Template



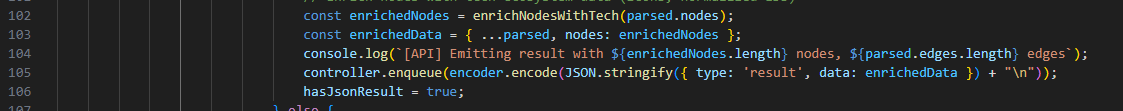
JSON Parsing



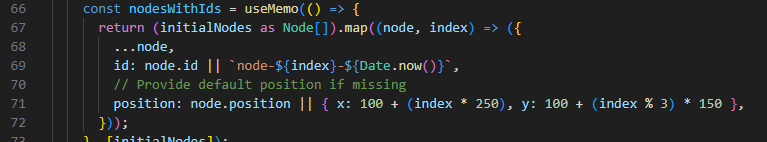




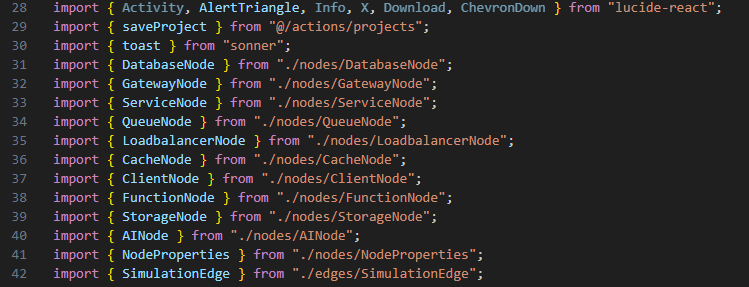




Rendering







Tier-Based Usage Controls

Plan quotas (current application runtime):

* free / Doodle - 30 generations per day
* starter / Sketch - 50 generations per day
* pro / Blueprint - 1000 generations per day

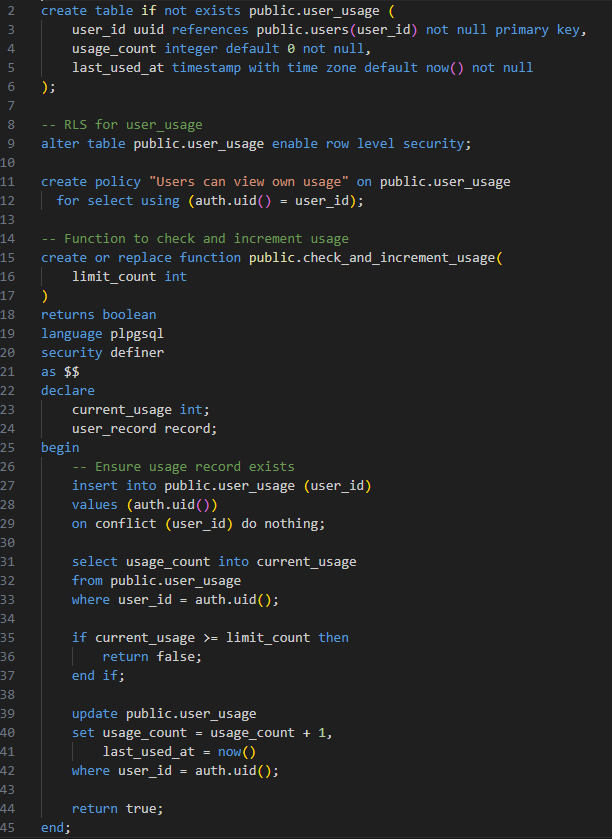
Rate limiting check



Subscription plans



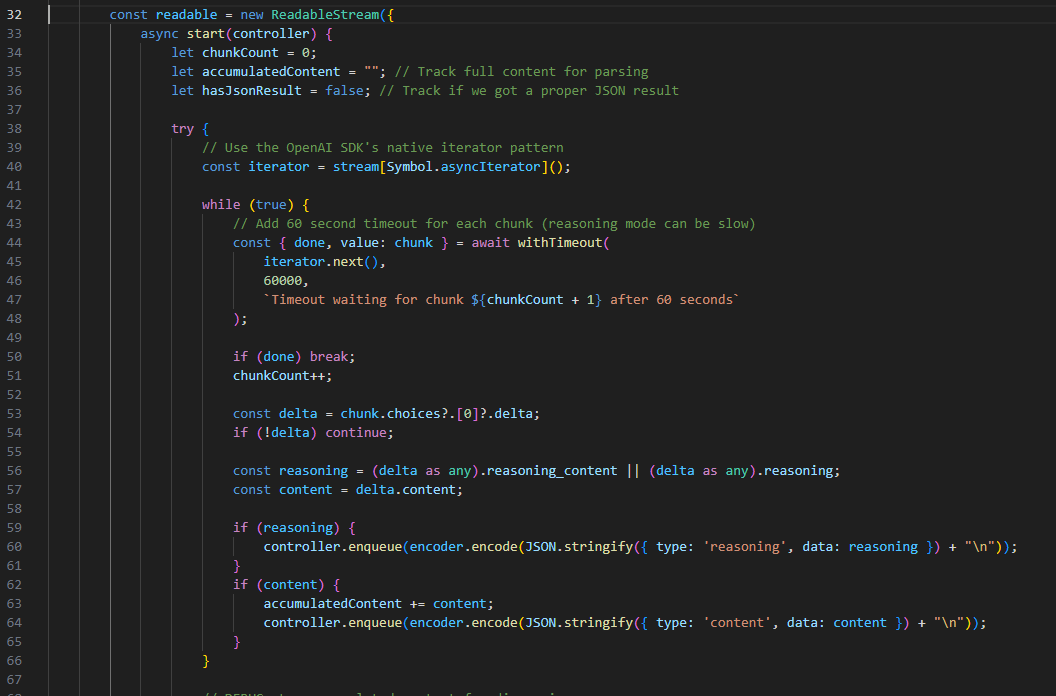
Database Schema

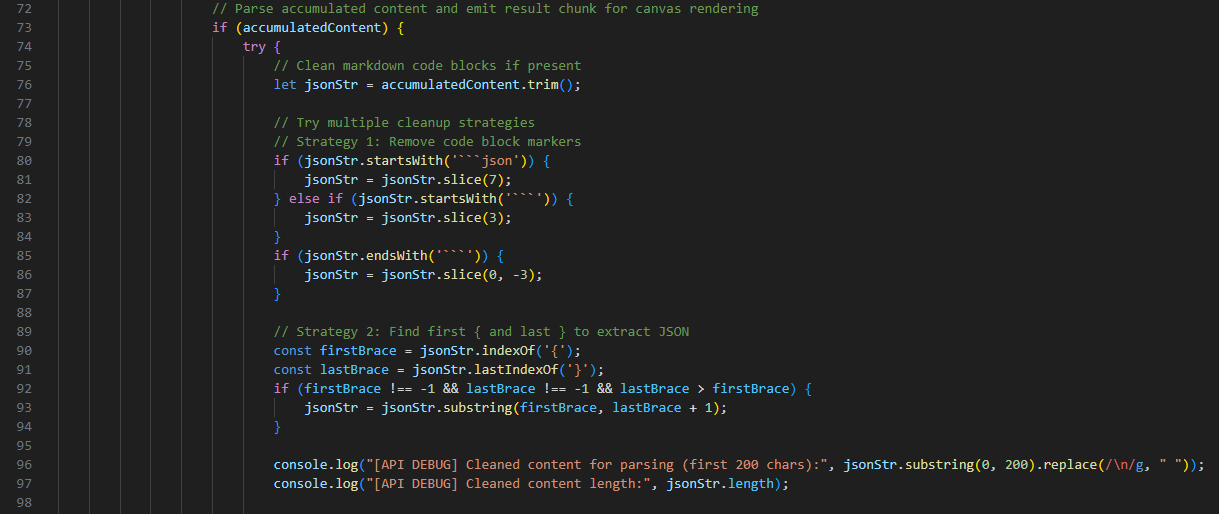


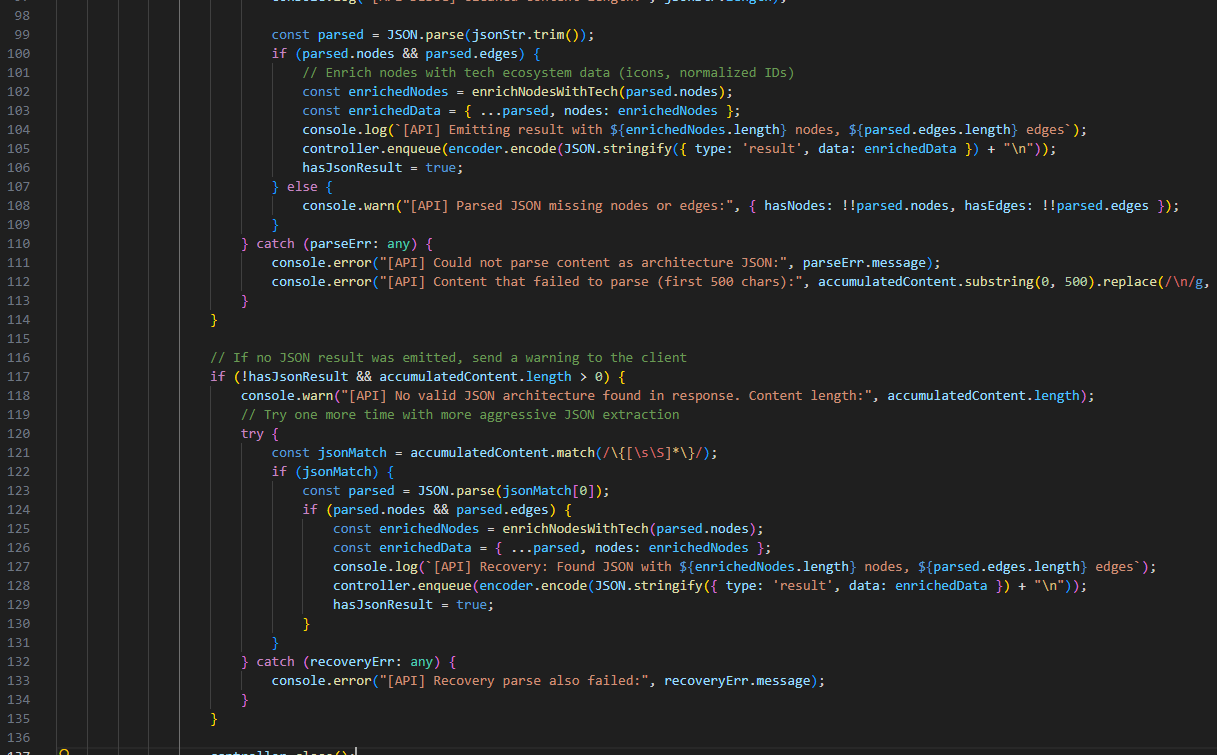
Streaming Response

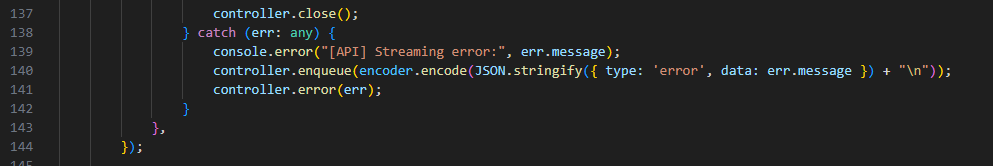
**Types:**

* reasoning – AI's thinking process
* content – Accumulating JSON chunks
* result – Final parsed + enriched architecture









# Application Development and Testing

Implementation Details

## Frontend

## 

#### User Authentication and Registration

The figure display the OAuth-based authentication system designed for Simulark-AI.

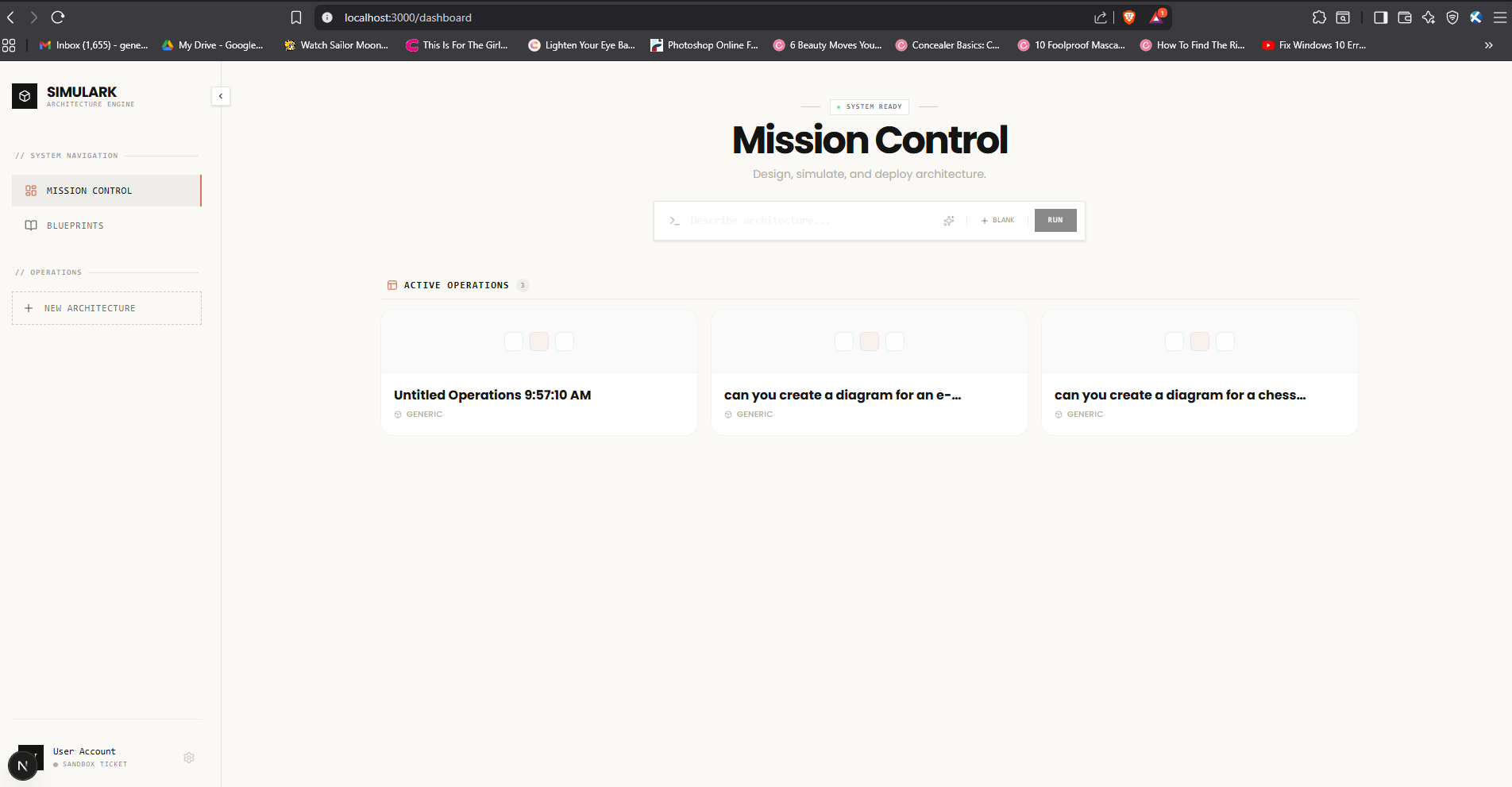
* **GitHub Login**: Dark button with GitHub icon for developers who prefer using their GitHub account
* **Google Login**: White button with Google icon for users with Google accounts
* **Error Banner**: Displays red error messages when authentication fails
* **Loading States**: Buttons show spinning icons during OAuth redirect process
* **Secure Access Divider**: Visual separator indicating secure authentication

**Authentication Flow**

The authentication system uses Supabase as the backend service:

1. User selects their preferred OAuth provider (GitHub or Google)
2. The app initiates the OAuth flow via Supabase Auth client
3. User is redirected to the selected provider for authentication
4. Upon successful authentication, the provider redirects back to
5. The callback route exchanges the authorization code for a session token
6. Session cookies are securely stored server-side
7. User is redirected to /dashboard on success
8. Authentication errors redirect to /auth/auth-code-error for user feedback

Dashboard Page



1. [Sidebar](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/layout/Sidebar.tsx:46) - Collapsible navigation with:
   * **System Navigation**: Mission Control (dashboard), Blueprints (templates)
   * **Operations**: New Architecture quick action
   * **User Menu**: Account info, subscription plan, settings
2. [DashboardPage](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/dashboard/page.tsx:29) - Main content with two sections:

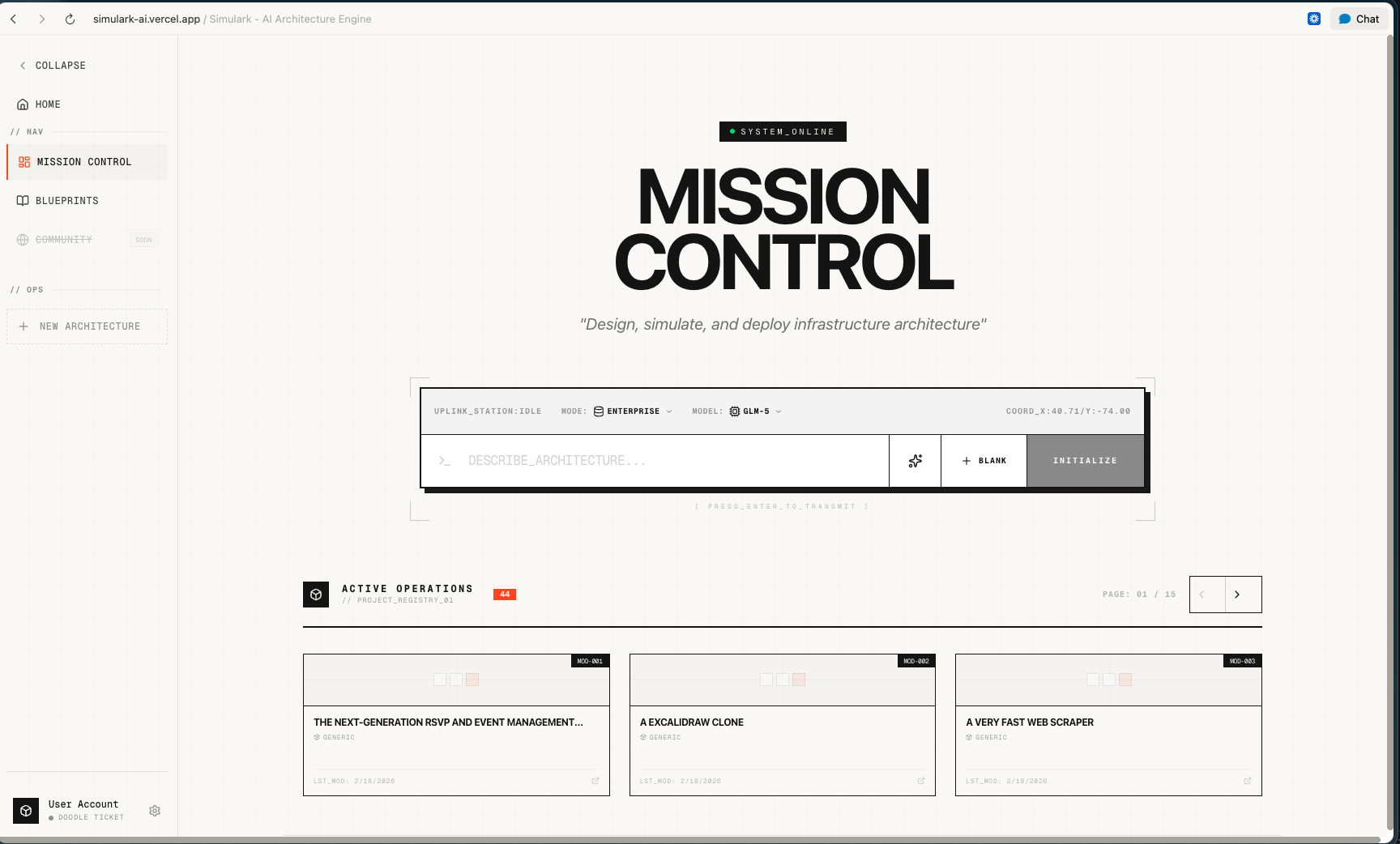
**Hero/Command Section**:

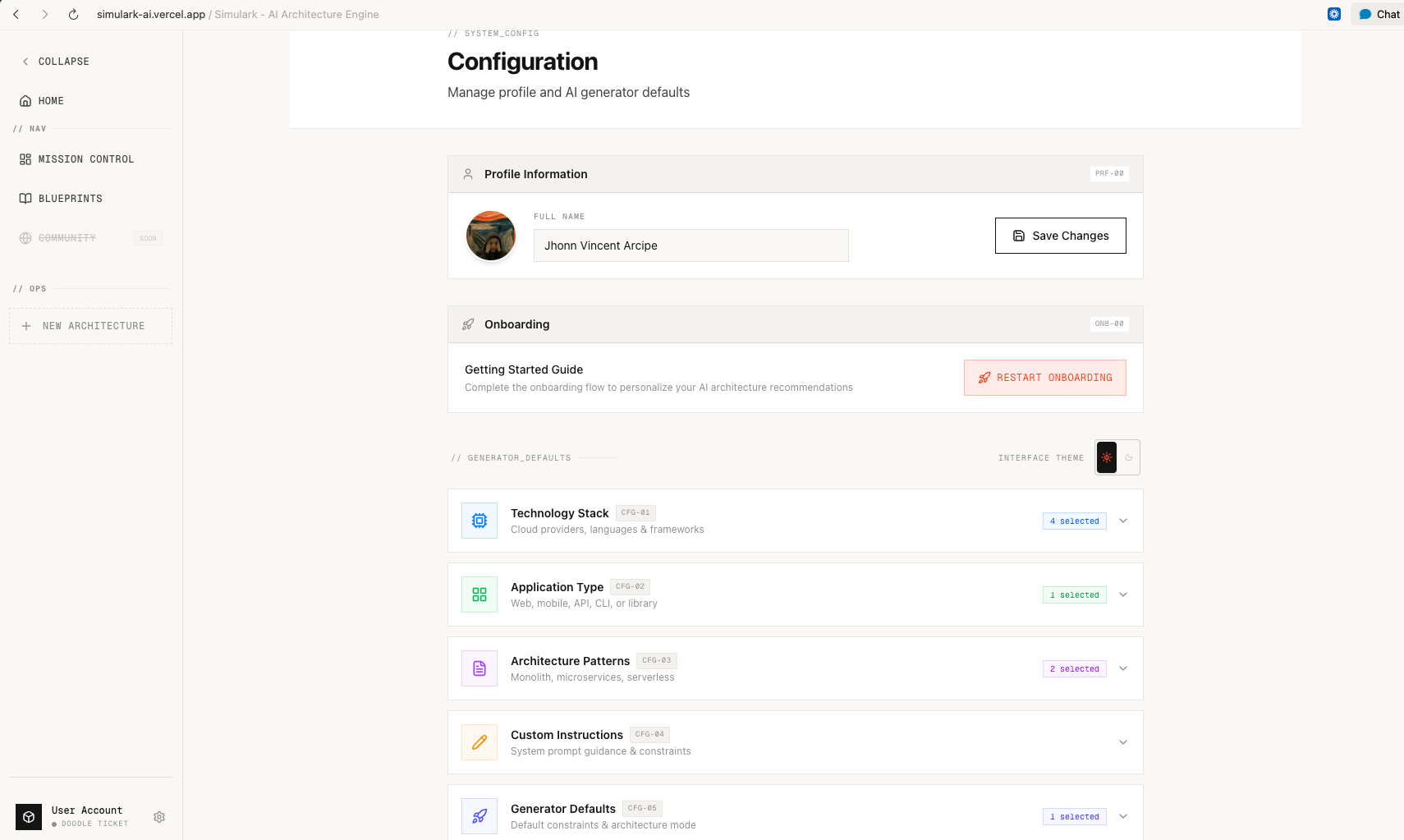
* + "System Ready" status indicator with pulsing green dot
  + Terminal-styled command input with corner markers
  + Three actions:
    - **Sparkles** - Random prompt generator from 10 architecture examples
    - **Blank** - Create empty project
    - **Run** - Submit natural language description to AI

**Active Operations Section**:

* + - Paginated grid of project cards (3 per page)
    - Shows: project name, provider type, last updated
    - Empty state with "No operations" message

Project Workspace Page





Key Components

1. [**WorkstationHeader**](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/projects/%5Bid%5D/page.tsx:31) - Top navigation with:
   * **Back button** to return to dashboard
   * **Project identity**: name, draft badge
   * **Menu bar**: File, View, Layout, Simulation dropdowns
   * **Status**: Saving indicator, latency
   * **Actions**: Deploy button, Share button
2. [ToolRail](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/projects/%5Bid%5D/page.tsx:186) - Left sidebar with:
   * Selection tools: Select, Pan, Slice
   * Creation tools: Node, Text, Image
   * Settings
3. [FlowEditor](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/canvas/FlowEditor.tsx:60) - Main canvas using ReactFlow:
   * **Node Types**: Database, Gateway, Service, Queue, Loadbalancer, Cache, Client, Function, Storage, AI
   * **Custom Edges**: SimulationEdge with congestion visualization
   * **Auto-save**: Debounced (1.5s) persistence
   * **Auto-layout**: Vertical (TB) or Horizontal (LR)
   * **Export**: Mermaid, PNG, SVG, PDF
   * **Chaos Mode**: Simulation of node failures
4. [AIAssistantPanel](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/canvas/AIAssistantPanel.tsx) - Right collapsible panel:
   * Thinking panel showing AI reasoning
   * Chat/command interface for generating architecture
   * Initial prompt passed from dashboard

#### Prompt request

#### 

**1. Starting Point** The user enters a description like "Build an e-commerce platform with Stripe and Redis" and hits send.

**2. Message Saving** The user's message is immediately saved to the database under their chat history for this project.

**3. API Request** The frontend sends the prompt to the /api/generate endpoint along with:

* The current state of the canvas (existing nodes and connections)
* The selected mode (default, startup, or corporate)
* The AI model to use

**4. Authentication & Rate Limits** The server checks if the user is logged in and if they have remaining generations for the day based on their subscription plan.

**5. AI Processing** The system constructs a detailed prompt for the AI that includes:

* A persona based on the selected mode (startup = lean CTO, corporate = enterprise architect, default = fullstack architect)
* The existing architecture if any (so the AI can modify it instead of starting fresh)
* Instructions on layering (frontend → backend → database)

**6. Streaming Response** The AI responds with a streaming output. Three things happen simultaneously:

* **Reasoning**: The AI's thinking process is shown in the Thinking Panel
* **Content**: Any additional text appears in the chat
* **Result**: The actual architecture JSON (nodes and edges) is sent for the canvas

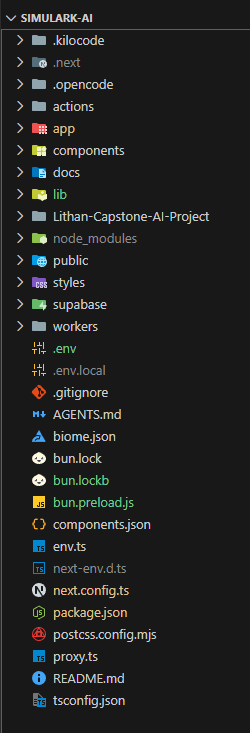
**7. Canvas Update** When the result arrives, the FlowEditor component receives the new nodes and edges and automatically:

* Adds them to the canvas
* Applies default positions if needed
* Auto-saves to the database after 1.5 seconds of no changes

**8. Summary Generation** After the architecture is generated, the system creates a nice markdown summary showing what was created (services, databases, queues, technologies used) and displays it in the chat.

The whole experience feels like talking to an architect who instantly draws your system on a whiteboard while explaining their thought process.

**File Structure**



[**app/**](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/) - Next.js App Router

* **Root layout & pages:**
  + [app/layout.tsx](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/layout.tsx) - Root layout with providers
  + [app/page.tsx](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/page.tsx) - Landing page
  + [app/globals.css](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/globals.css) - Global Tailwind styles
* **Authentication:**
  + [app/auth/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/auth/) - Auth pages (signin, callback, error)
* **Dashboard:**
  + [app/dashboard/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/dashboard/) - Protected dashboard with settings, templates
* **API Routes:**
  + [app/api/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/api/) - REST endpoints for chats, projects, context, admin subscriptions
* **Projects:**
  + [app/projects/[id]/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/app/projects/%5Bid%5D/) - Individual project editor

[**actions/**](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/) - Server Actions

* [actions/ai-orchestrator.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/ai-orchestrator.ts) - AI generation orchestration
* [actions/chats.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/chats.ts) - Chat operations
* [actions/projects.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts) - Project CRUD

[**components/**](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/) - React Components

* [components/canvas/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/canvas/) - Flow editor components:
  + [FlowEditor.tsx](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/canvas/FlowEditor.tsx) - Main canvas
  + [AIAssistantPanel.tsx](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/canvas/AIAssistantPanel.tsx) - AI chat panel
  + [nodes/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/canvas/nodes/) - Node types (Service, Database, Cache, etc.)
  + [edges/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/canvas/edges/) - Edge types with labels
* [components/layout/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/layout/) - Layout components:
  + [Sidebar.tsx](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/layout/Sidebar.tsx) - Navigation sidebar
  + [DashboardLayout.tsx](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/layout/DashboardLayout.tsx) - Dashboard wrapper
* [components/auth/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/auth/) - Auth UI components
* [components/dashboard/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/dashboard/) - Dashboard-specific components
* [components/marketing/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/marketing/) - Marketing pages (Hero, CTA, etc.)
* [components/ui/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/ui/) - Shadcn/UI primitives
* [components/subscription/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/components/subscription/) - Subscription/upgrades

[**lib/**](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/) - Utilities & Core Logic

* **AI & Generation:**
  + [lib/ai-client.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/ai-client.ts) - AI provider abstraction (OpenAI, ZhipuAI, OpenRouter, Kimi)
  + [lib/prompt-engineering.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/prompt-engineering.ts) - AI prompts
  + [lib/terraform-generator.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/terraform-generator.ts) - Terraform output
  + [lib/skill-generator.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/skill-generator.ts) - Skill code generation
* **Data & State:**
  + [lib/store.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/store.ts) - Zustand state management
  + [lib/subscription.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/subscription.ts) - Subscription tier logic
  + [lib/templates.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/templates.ts) - Project templates
* **Canvas:**
  + [lib/node-schemas.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/node-schemas.ts) - Node type definitions
  + [lib/tech-normalizer.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/tech-normalizer.ts) - Technology normalization
* **Infrastructure:**
  + [lib/circuit-breaker.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/circuit-breaker.ts) - Resilience pattern
  + [lib/rate-limit.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/rate-limit.ts) - Rate limiting
  + [lib/redis.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/redis.ts) - Redis client

[supabase/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/)

* [supabase/migrations/](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/) - Database migrations

[**docs/**](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/docs/)

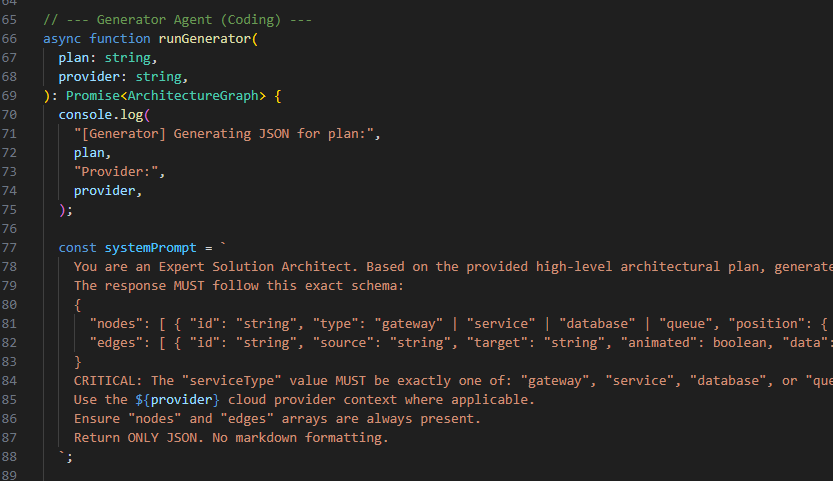
* Design guides, checklists, and reference documentation

**Configuration Files**

* [package.json](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/package.json) - Dependencies (Bun runtime)
* [next.config.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/next.config.ts) - Next.js configuration
* [biome.json](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/biome.json) - Code formatting/linting
* [env.ts](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/env.ts) - Environment variable validation
* [tsconfig.json](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/tsconfig.json) - TypeScript configuration

## Backend

## 

****

AI Orchestrator

1. runAggregator (Thinking Agent) - [actions/ai-orchestrator.ts:39](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/ai-orchestrator.ts:39)

async function runAggregator(prompt: string): Promise<string>

**Purpose:** Analyzes user request and creates a high-level architectural plan.

**How it works:**

* Takes the user's natural language prompt
* Sends to **Upstage Solar Pro 3** model via OpenRouter
* System prompt (line 48-50) instructs it to:
  + Act as an "Expert System Architect"
  + Outline components (Gateway, Service, Database, Queue)
  + Focus on connections between components
  + Output **plain text** (not JSON)

2. runGenerator (Coding Agent) - [actions/ai-orchestrator.ts:66](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/ai-orchestrator.ts:66)

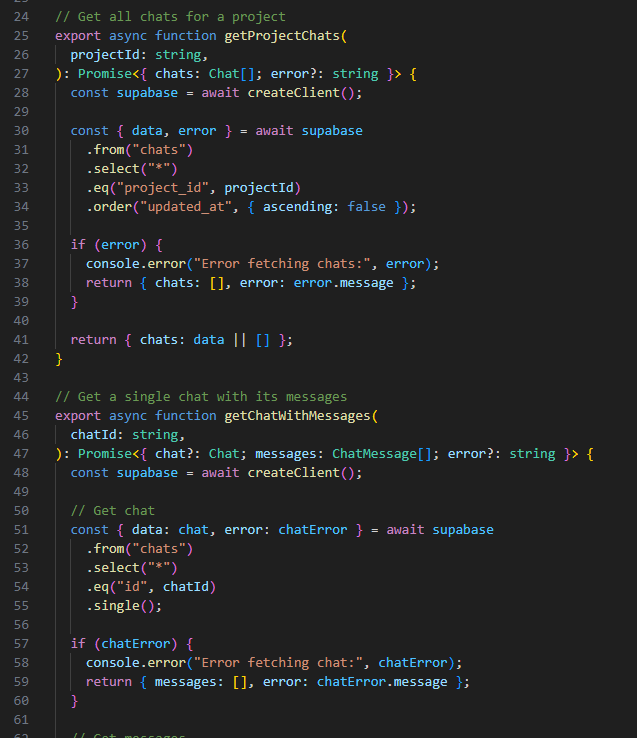
async function runGenerator(plan: string, provider: string): Promise<ArchitectureGraph>

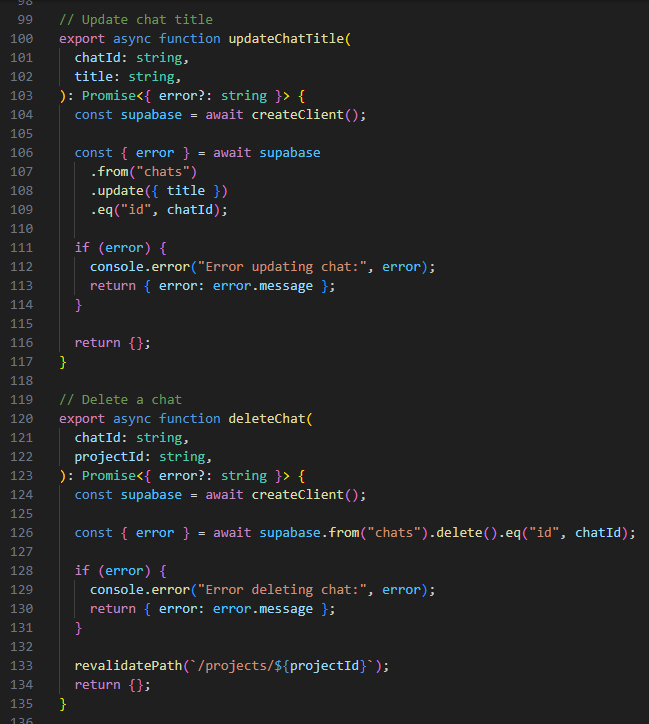
**Purpose:** Converts the aggregator's plan into structured JSON for the canvas.

**How it works:**

1. Takes the aggregator's plan + cloud provider context (AWS/GCP/Azure)
2. Sends to **Mistral Small 3.1** model (with Gemma fallback)
3. System prompt (line 77-88) enforces strict JSON schema:
   * nodes: Array with id, type, position, data
   * edges: Array with source, target, animated, data
4. Validates output against [ArchitectureGraphSchema](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/lib/schema/graph.ts)

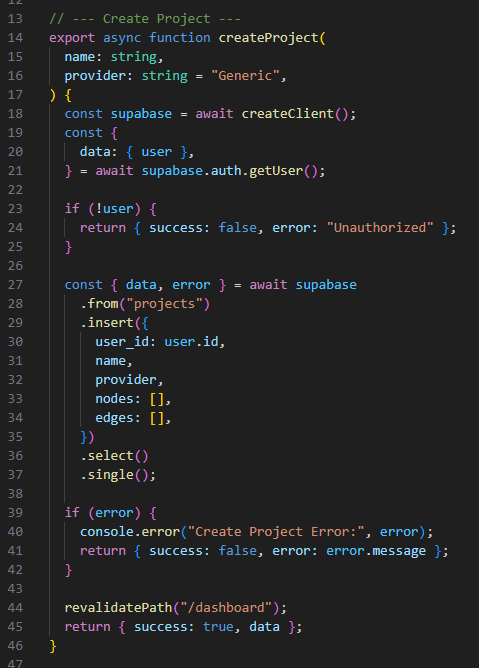
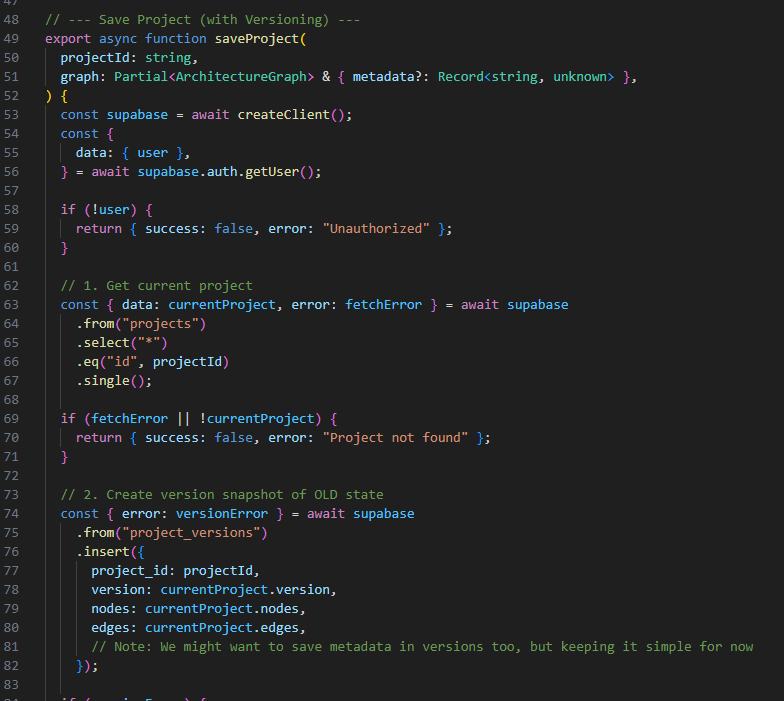
Chat System

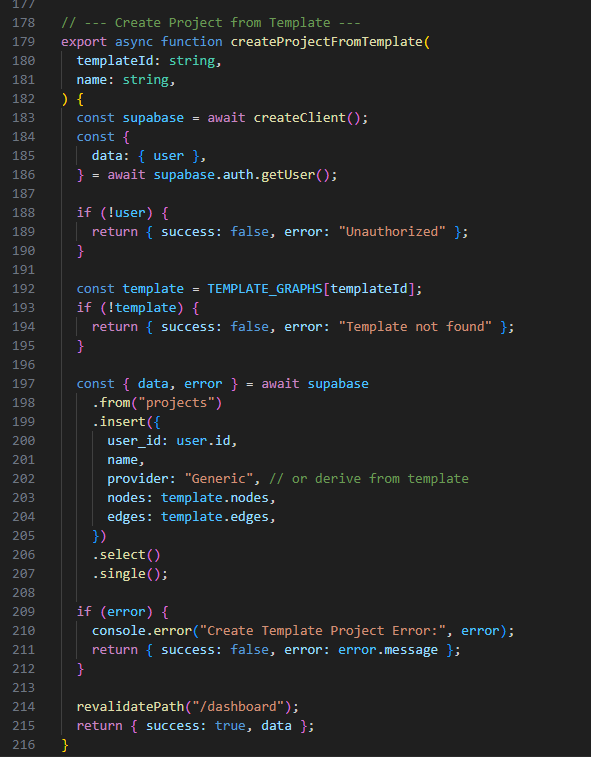
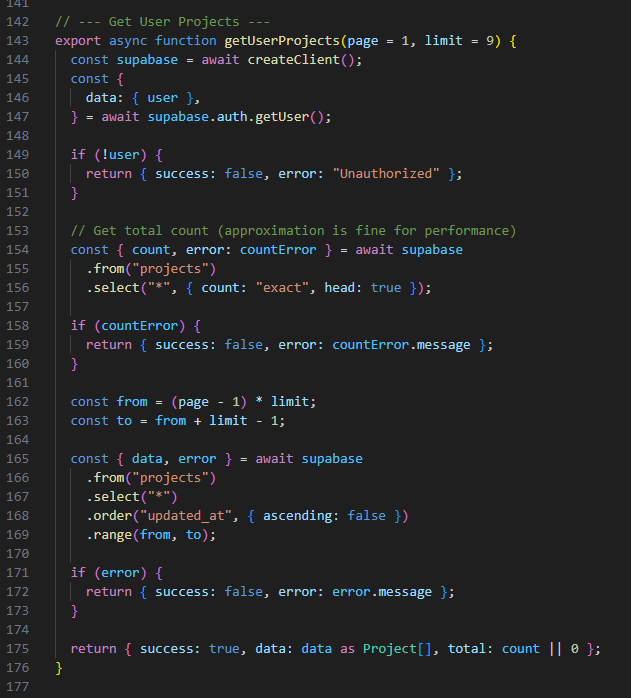
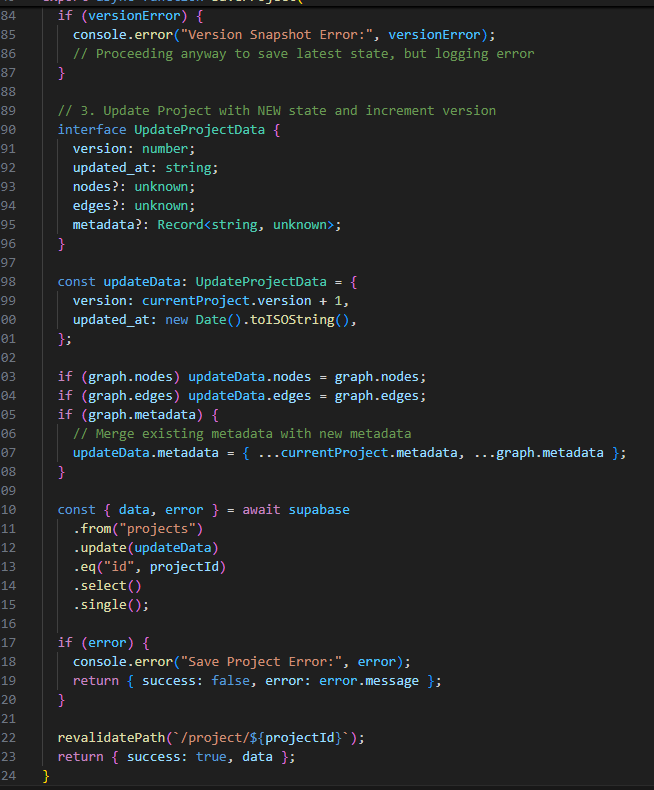
****



1. **Auto timestamp update** ([actions/chats.ts:158-162](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/chats.ts:158)): When adding messages, the chat's updated\_at is refreshed to sort by recent activity.
2. **Cascade delete** ([actions/chats.ts:126](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/chats.ts:126)): Deleting a chat should cascade to chat\_messages via Supabase RLS policy.
3. **Revalidation** ([actions/chats.ts:95](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/chats.ts:95), [actions/chats.ts:133](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/chats.ts:133)): After create/delete, calls revalidatePath() to refresh the UI.
4. **Reasoning support** ([actions/chats.ts:142](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/chats.ts:142)): Messages can store reasoning field for AI thinking process display.

Projects





**Purpose:** Server actions for managing architecture diagram projects

Functions

* [createProject()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:14) - Creates new empty project with name, provider, empty nodes/edges
* [saveProject()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:49) - Saves project with auto-versioning (creates snapshot before updating)
* [getProject()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:127) - Fetches single project by ID
* [getUserProjects()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:143) - Returns paginated list of user's projects (sorted by updated\_at)
* [createProjectFromTemplate()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:179) - Creates project from predefined template (e.g., "Microservices", "Serverless")

Key Feature: Versioning

* Before updating, old state is saved to project\_versions table
* Version number increments on each save
* Enables history tracking and version restoration

Database Tables

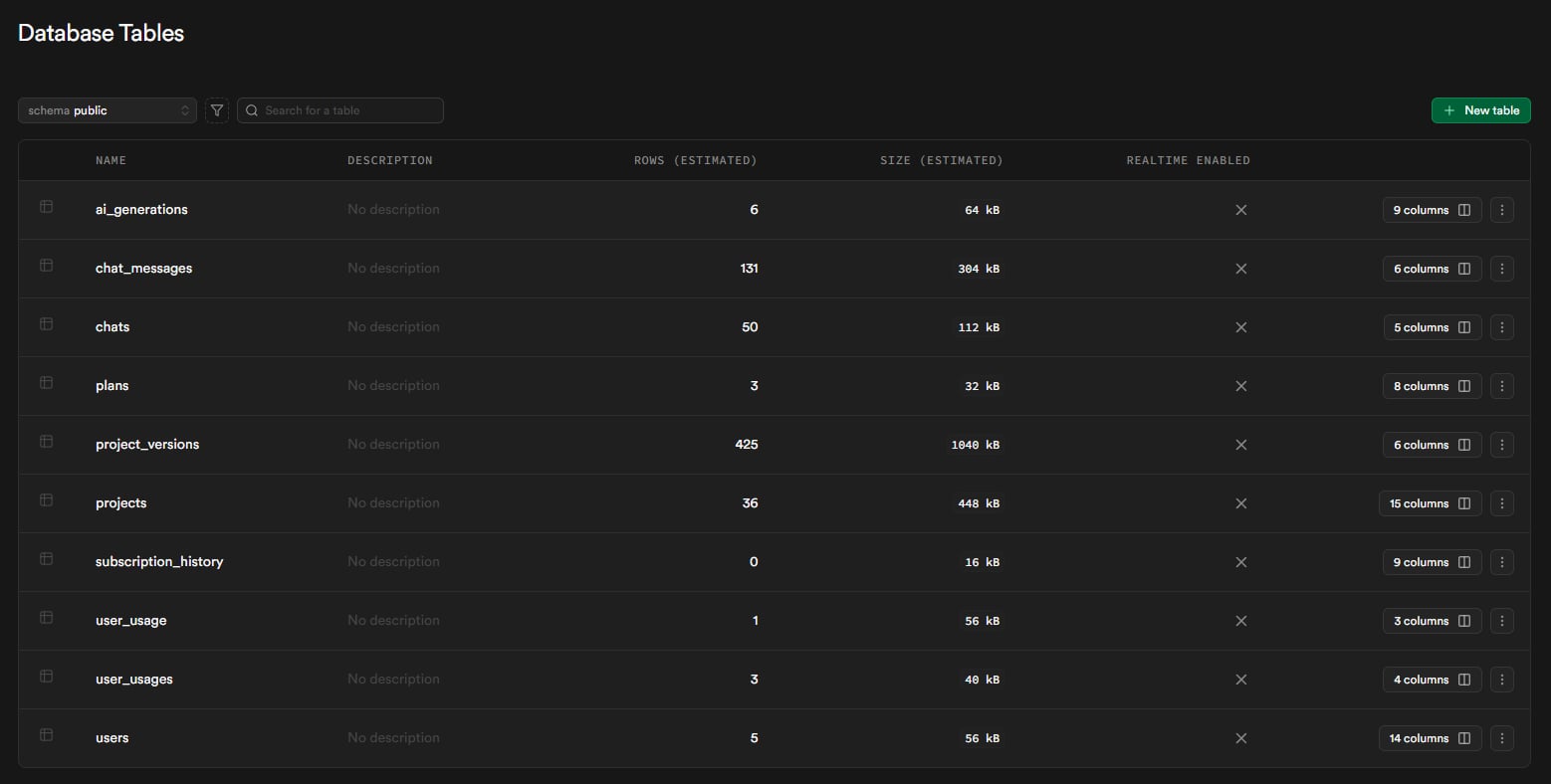
* projects - Main project storage (nodes, edges, metadata, version)
* project\_versions - Snapshots of previous states

Revalidation

* [createProject()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:44) - Revalidates /dashboard
* [createProjectFromTemplate()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:214) - Revalidates /dashboard
* [saveProject()](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/actions/projects.ts:122) - Revalidates /project/{id}

## Database Implementation

For backend persistence, Simulark uses Supabase (PostgreSQL) for authentication, project storage, chat history, usage tracking, and subscription lifecycle data. Upstash Redis is integrated for caching and rate-limit related workflows. Database access is protected with Row-Level Security and scoped policies.



Core Database Tables

* **users**
  + Extends Supabase Auth for user profiles
  + Fields: user\_id, email, full\_name, avatar\_url, subscription\_tier, subscription\_status, subscription\_expires\_at, tier\_started\_at, manual\_override, stripe\_customer\_id, is\_admin, onboarding\_completed, onboarding\_step, onboarding\_skipped, onboarding\_data, preferences, created\_at, updated\_at
  + Auto-synced via trigger when new auth user is created
* **projects**
  + Stores architecture diagram projects
  + Fields: id, user\_id, name, description, provider, nodes, edges, metadata, is\_public, version, share\_token, deleted\_at, created\_at, updated\_at
  + Supports JSON storage for nodes and edges (React Flow format)
  + Soft delete via deleted\_at timestamp
* **project\_versions**
  + Version history snapshots for projects
  + Fields: id, project\_id, version, nodes, edges
  + Created automatically before each project update
  + Enables history tracking and rollback capability
* ai\_generations
  + Stores generation telemetry and request metadata for auditability and diagnostics
  + Fields: id, user\_id, project\_id, prompt, model\_aggregator, model\_generator, success, tokens\_used
  + Used for analytics and debugging
* user\_usages (application runtime usage table)
  + Primary daily generation tracking table used by server-side checkRateLimit() logic
  + Fields: user\_id, generation\_count, date, last\_updated (user\_id references auth.users)
  + Updated on each generation request by server-side rate-limit checks
* **chats**
  + Chat sessions per project
  + Fields: id, project\_id, title, created\_at, updated\_at
  + Multiple chats can exist per project

subscription\_history

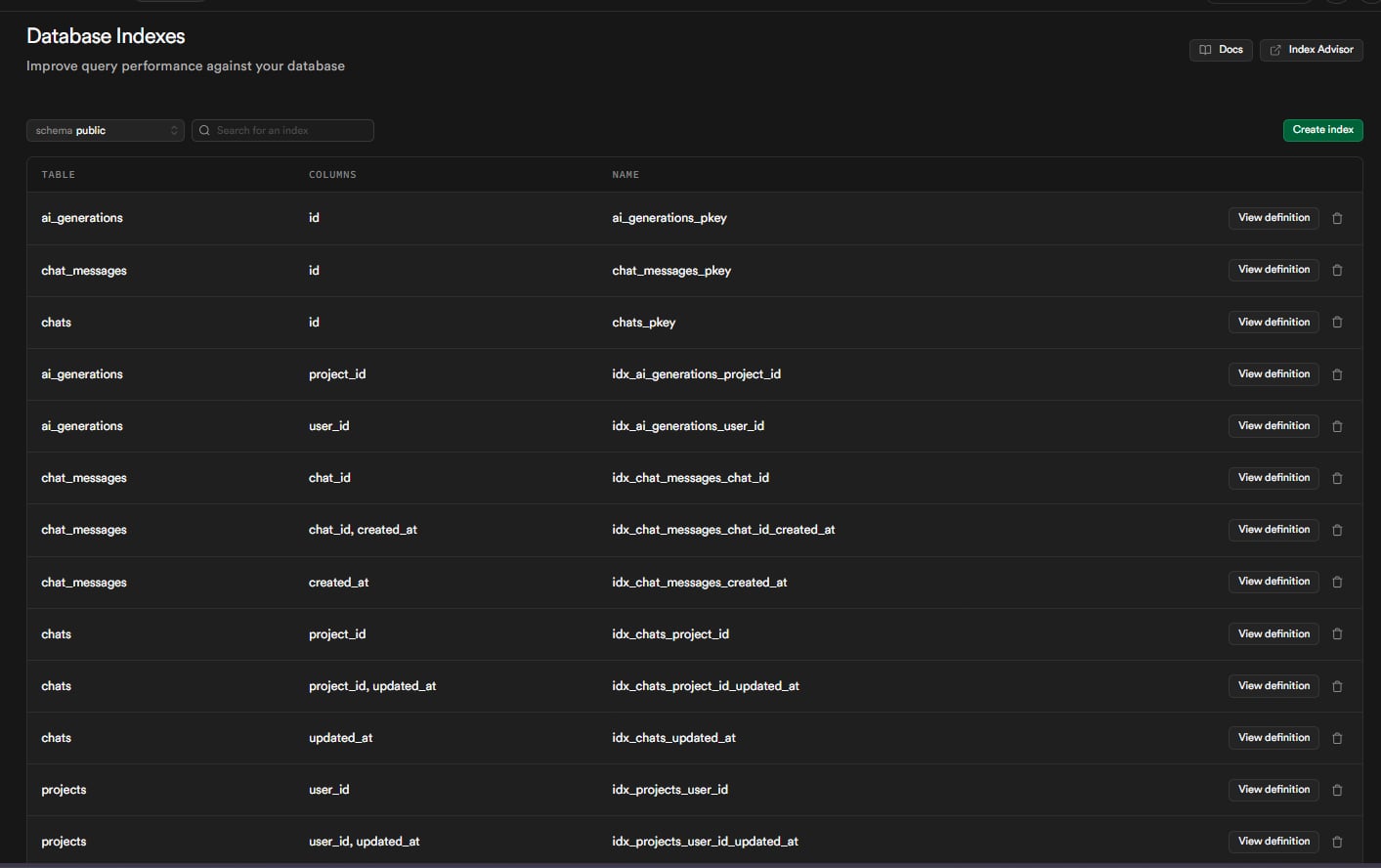
Subscription audit trail table

Fields: id, user\_id, old\_tier, new\_tier, old\_status, new\_status, changed\_by, change\_reason, created\_at

Used by admin lifecycle operations and downgrade/upgrade history

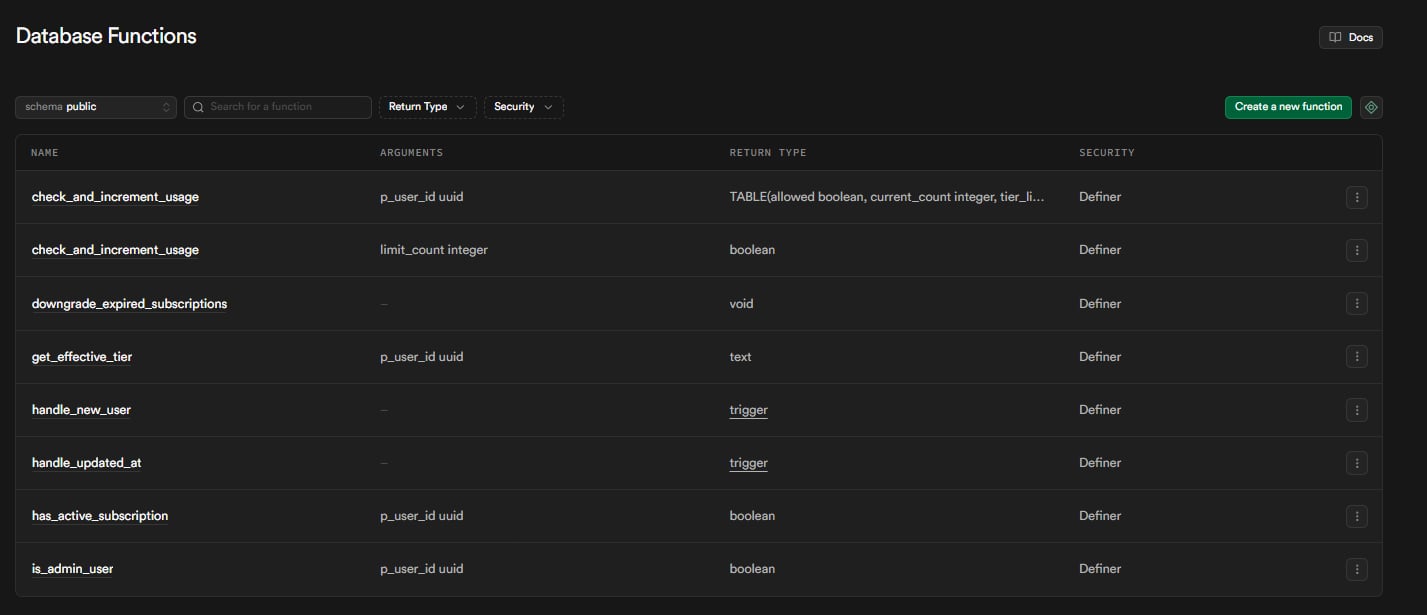
RLS allows user self-view and admin management policies

* **chat\_messages**
  + Individual messages within chats
  + Fields: id, chat\_id, role, content, reasoning, created\_at
  + Cascades delete when parent chat is deleted
  + Supports AI reasoning content storage
* user\_usage (legacy / RPC rate-limit path)
  + Legacy counter table referenced by SQL function check\_and\_increment\_usage() in migration pathway
  + Fields: user\_id, usage\_count, last\_used\_at
  + Retained for compatibility with earlier migration/RPC strategy; active app subscriptions are normalized to free/starter/pro.



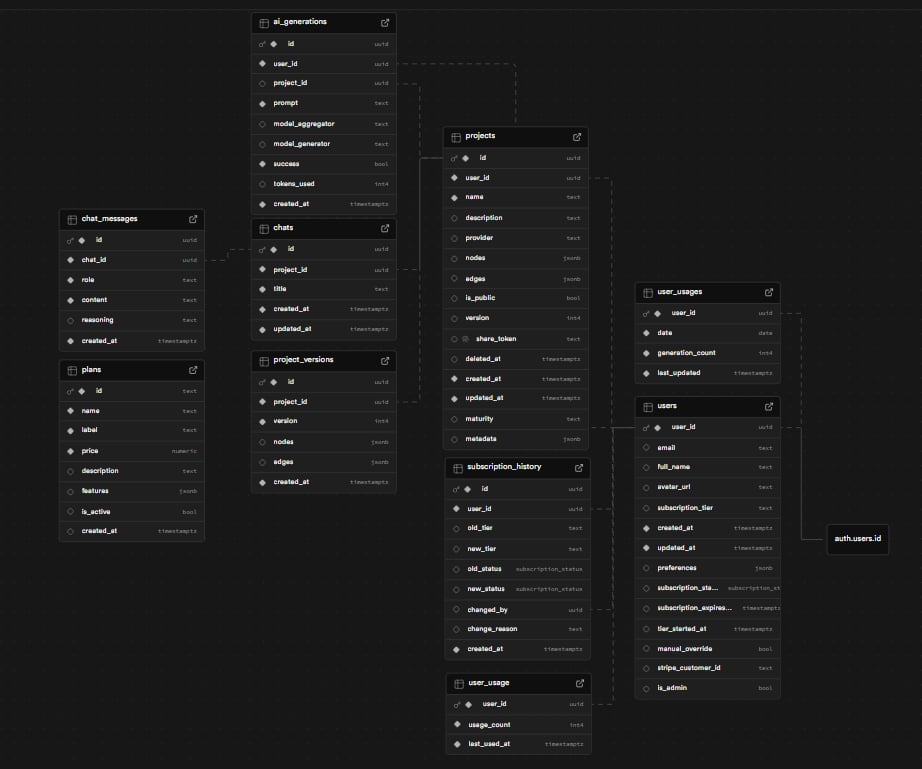
Database Indexes

* **idx\_chats\_project\_id**
  + On: public.chats(project\_id)
  + Purpose: Fast lookup of chats by project
* **idx\_chats\_updated\_at**
  + On: public.chats(updated\_at DESC)
  + Purpose: Sort chats by most recently updated
* **idx\_chat\_messages\_chat\_id**
  + On: public.chat\_messages(chat\_id)
  + Purpose: Fast retrieval of messages within a chat
* **idx\_chat\_messages\_created\_at**
  + On: public.chat\_messages(created\_at)
  + Purpose: Sort messages chronologically
* **idx\_users\_subscription\_expires**
  + On: public.users(subscription\_expires\_at)
  + Condition: WHERE subscription\_status = 'active'
  + Purpose: Optimize subscription expiry checks for billing
* **idx\_users\_admin**
  + On: public.users(is\_admin)
  + Condition: WHERE is\_admin = true
  + Purpose: Fast admin user lookups



Database Functions

* **handle\_updated\_at()**
  + Trigger function for auto-updating updated\_at timestamps
  + Called on BEFORE UPDATE for projects table
  + Security: DEFINER, SET search\_path = public
  + Location: [20260210\_security\_fixes.sql](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/20260210_security_fixes.sql:5)
* **handle\_new\_user()**
  + Trigger function for auto-creating user profile on auth signup
  + Called on AFTER INSERT for auth.users
  + Inserts into public.users with email and metadata
  + Location: [20260131\_init\_schema.sql](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/20260131_init_schema.sql:17)
* **check\_and\_increment\_usage(p\_user\_id uuid)**
  + Returns TABLE(allowed boolean, current\_count integer, tier\_limit integer)
  + SQL-level rate limiting function (legacy/RPC pathway; tier limits defined in migration function body)
  + Creates usage record when missing and increments atomically when allowed
  + Security: DEFINER, SET search\_path = public
  + Location: [20260210\_security\_fixes.sql](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/20260210_security_fixes.sql:18)
* **has\_active\_subscription(p\_user\_id uuid)**
  + Returns boolean
  + Checks subscription\_status and expiry date
  + Returns true for 'active' or 'trialing' with valid expiry
  + Location: [20260210\_enhanced\_subscriptions.sql](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/20260210_enhanced_subscriptions.sql:27)
* **downgrade\_expired\_subscriptions()**
  + Returns void
  + Auto-downgrades users with expired subscriptions to free tier
  + Respects manual\_override flag
  + Security: DEFINER
  + Location: [20260210\_enhanced\_subscriptions.sql](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/20260210_enhanced_subscriptions.sql:49)
* **get\_effective\_tier(p\_user\_id uuid)**
  + Returns text
  + Returns user's tier with validity check
  + Returns 'free' if subscription expired
  + Location: [20260210\_enhanced\_subscriptions.sql](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/20260210_enhanced_subscriptions.sql:66)
* **is\_admin\_user(p\_user\_id uuid)**
  + Returns boolean
  + Security definer function for admin checks
  + Bypasses RLS to prevent infinite recursion
  + Location: [20260210\_fix\_rls\_recursion.sql](vscode-webview://1pv5ft6iln941871d9v79bo4l9tch3p8j2p268lebte5qeanbdhj/supabase/migrations/20260210_fix_rls_recursion.sql:13)

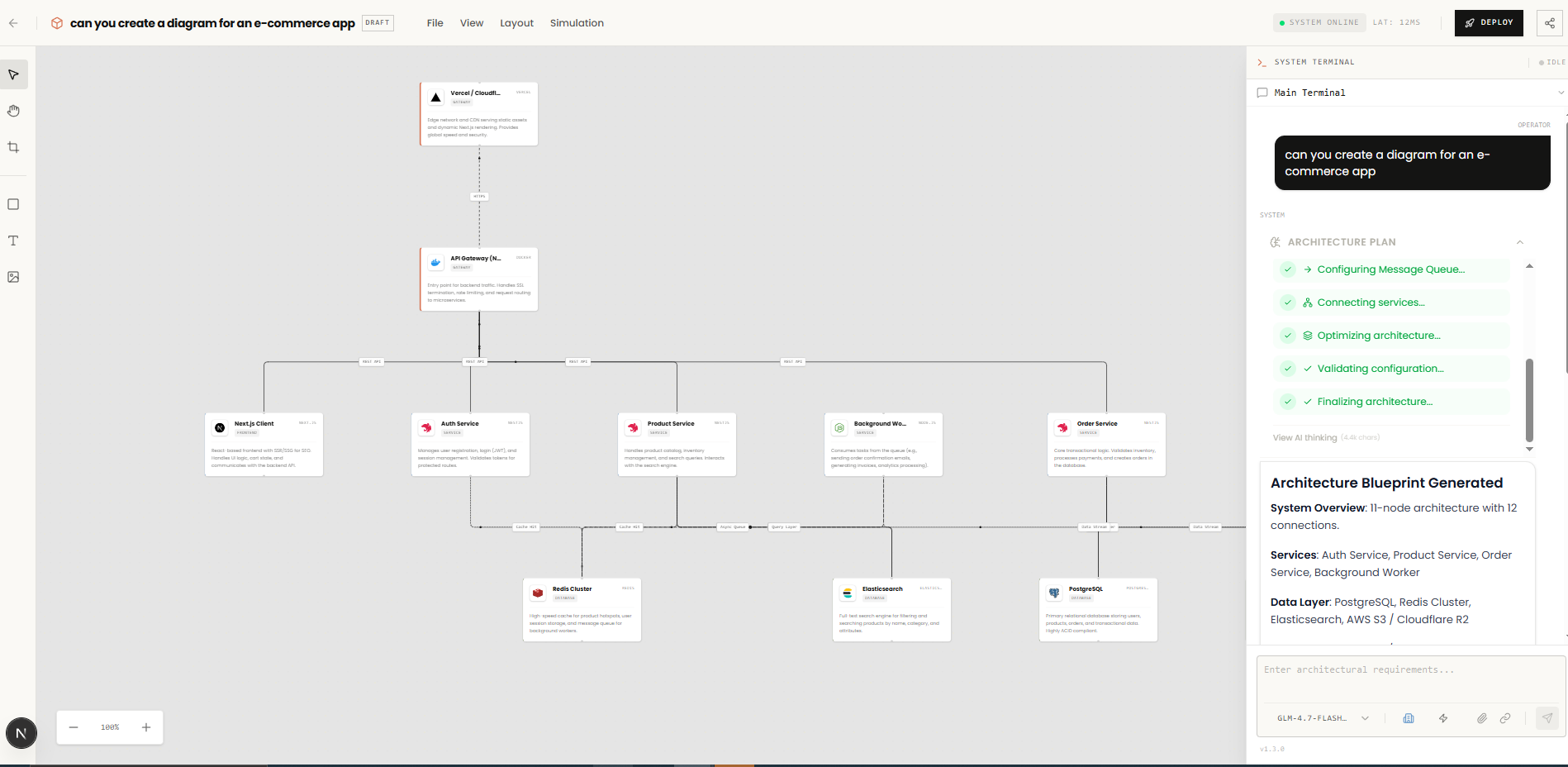




### Testing Methodology

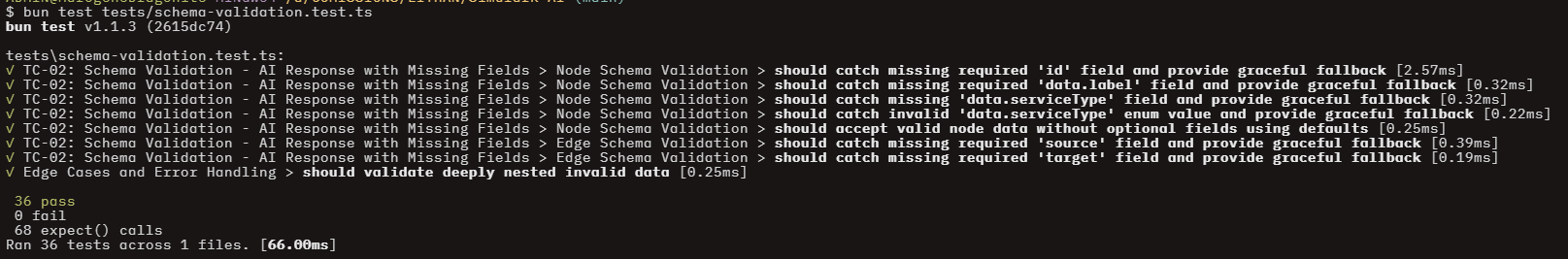
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Feature** | **Test Steps** | **Expected Results** | **Status** |
| TC-01 | AI Generation | Enter prompt “Create an e-commerce app diagram". Click Generate. | Returns valid graph with Gateway, Service, Database, Queue nodes | PASS |
| TC-02 | Schema Validation | Mock AI response with missing fields | Validation catches error, graceful fallback | PASS |
| TC-03 | Canvas Interaction | Drag nodes, connect edges | Edges animate, referential integrity maintained | PASS |
| TC-04 | Rate Limiting | Exceed generation limit | Returns 429 with Retry-After header | PASS |
| TC-05 | Chaos Mode | Enable Chaos Mode, kill a node | Traffic reroutes, visual feedback | PASS |
| TC-06 | Skill Export | Click "Export Skill" | Downloads SKILL.md with architecture rules | PASS |
| TC-07 | Project Versioning | Edit and save project | New version snapshot created | PASS |

#### Evidence:



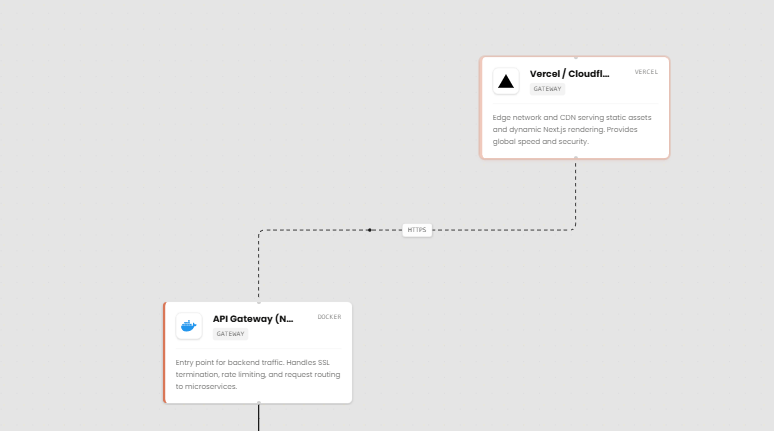
TC-01

Created a diagram for the prompt “can you create a diagram for an e-commerce app”

 TC-02

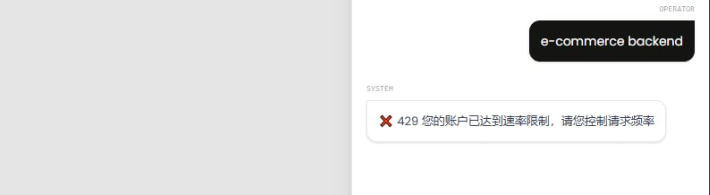
Tested missing fields using vitest and valibot



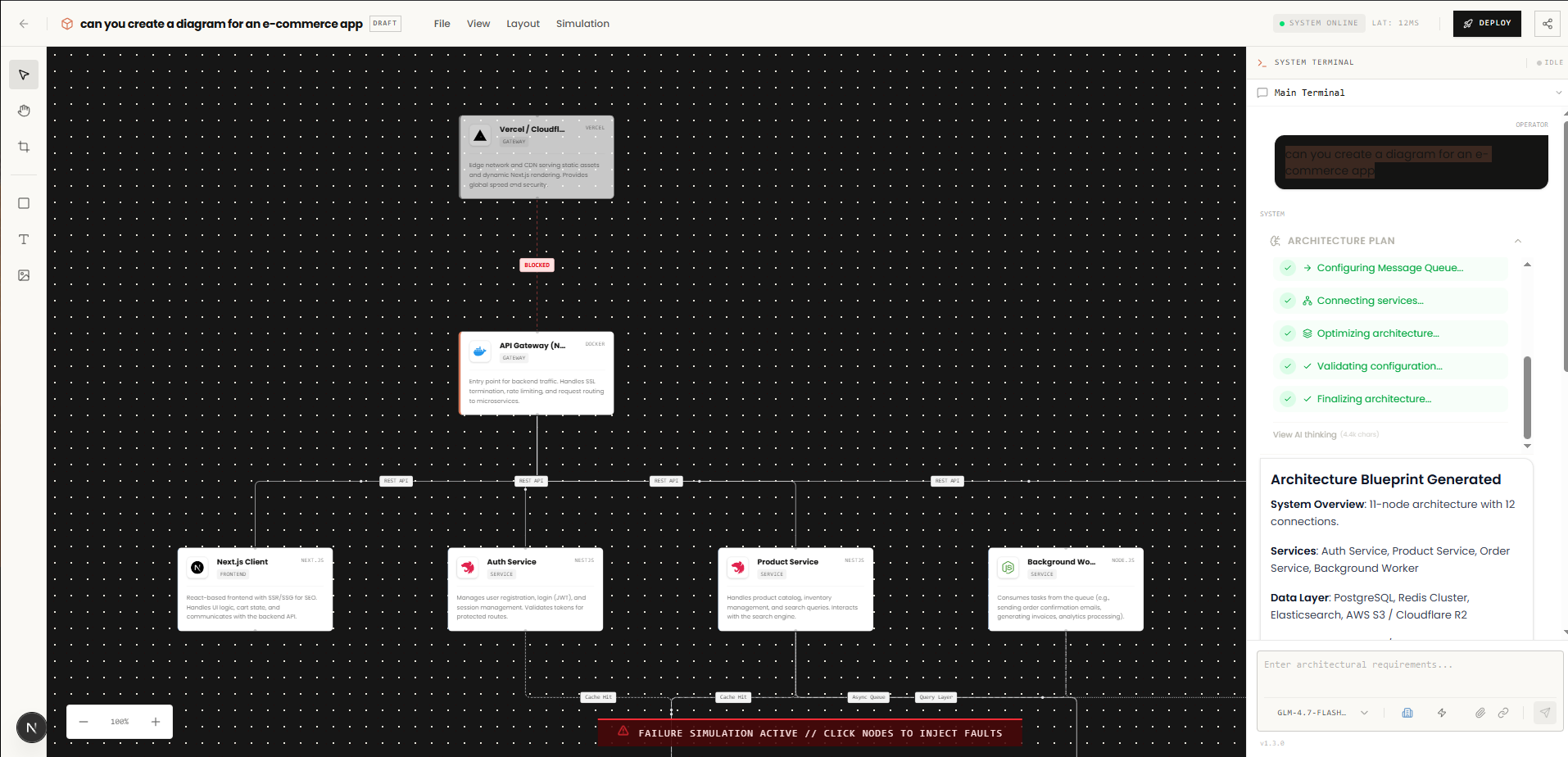


Can drag the figure and alter the connections between them

TC-04

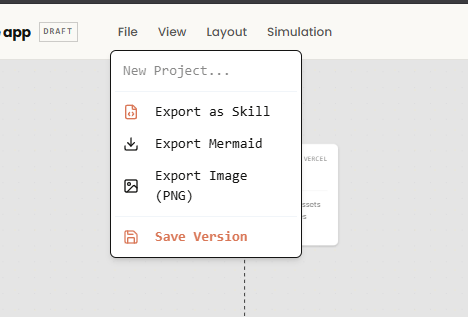


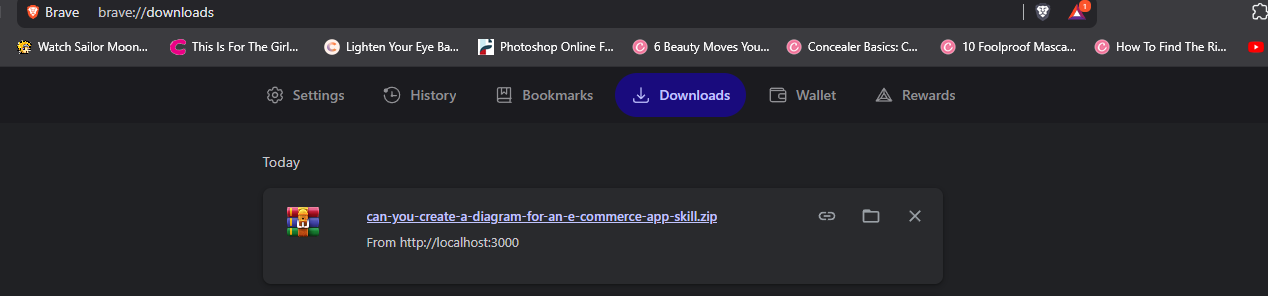
The user exceeded the limit of the free tier



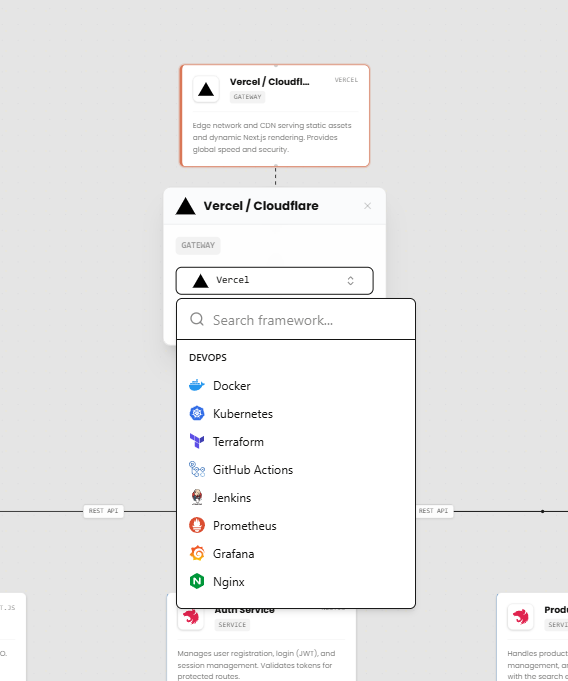
Traffic reroutes, visual feedback

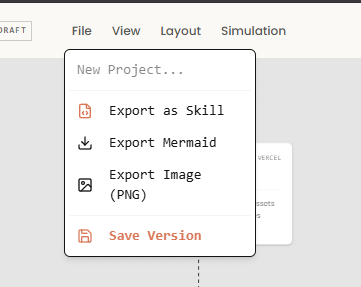
TC-06





Downloaded a skill files





User can edit and save a project

#### AI Model Validation

This section documents the scientific testing of the two AI modules integrated into the Simulark backend. The objective is to quantify the accuracy and reliability of the algorithms using the final dataset generated during the testing phase.

**AI Client Provider Integration**

* Objective: To verify that the AI client initializes provider configurations correctly, maps model IDs to the intended provider path, and preserves deterministic fallback behavior under failure scenarios.
* **Methodology:** A suite of automated unit tests was executed using a mock-driven strategy to validate configuration logic and error handling.
  + Scope: 18 test cases grouped by client initialization, provider configuration, fallback behavior, model mapping, and request setup.
  + Provider Coverage: The repository test suite covers Zhipu, OpenRouter, Kimi, Google, Minimax, Anthropic, and NVIDIA-routed model paths.
  + **Environment:** TypeScript-based test runner (ai-client.test.ts).
* **Metrics:** Execution Latency (ms) and Logical Pass Rate.

#### Test Results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Test Case** | **Status** | **Duration** |
| 1. createAIClient | Should create client for Zhipu provider | PASSED | 1.02ms |
|  | Should create client for OpenRouter provider | PASSED | 0.07ms |
|  | Should default to Zhipu provider | PASSED | 0.05ms |
|  | Should handle missing API key gracefully | PASSED | 0.06ms |
| 2. Provider Config | Should have correct Zhipu configuration | PASSED | 0.02ms |
|  | Should have correct OpenRouter configuration | PASSED | 0.07ms |
|  | Should have correct Kimi configuration | PASSED | 0.06ms |
|  | Should have correct Google configuration | PASSED | 0.02ms |
|  | Should have correct Minimax configuration | PASSED | 0.06ms |
|  | Should have correct Anthropic configuration | PASSED | 0.02ms |
|  | Should set timeout to 30 seconds | PASSED | 0.05ms |
| 3. Fallback Chain | Should have Zhipu as primary provider | PASSED | 0.06ms |
|  | Should have OpenRouter as fallback provider | PASSED | 0.03ms |
| 4. Model Mapping | Should map glm-4.7-flash to Zhipu provider | PASSED | 0.03ms |
|  | Should map DeepSeek models to OpenRouter provider | PASSED | 0.02ms |
|  | Should map kimi-k2.5 to Kimi provider | PASSED | 0.05ms |
| 5. Multi-Provider | Should support all AI providers | PASSED | 0.16ms |
|  | Should have unique models for each provider | PASSED | 0.10ms |
| 6. Client Headers | Should create client with proper configuration | PASSED | 0.02ms |

Analysis: AI client tests document expected routing and configuration behavior for the current multi-provider stack. Assertions verify base URLs, model defaults, timeout behavior, and fallback assumptions. Together these checks reduce regression risk when provider settings evolve.

**AI Resilience & Response Validation**

* **Objective:** To verify the system's ability to maintain stability through fault-tolerant retry logic, circuit breaker patterns, and strict structural validation of AI-generated content.
* **Methodology:** A stress-testing and fault-injection approach was applied to ensure the middleware can parse irregular AI outputs and recover from API-side failures.
  + **Dataset:** 28+ simulated failure and response scenarios.
  + **Key Logic:** Exponential backoff with jitter and "Nodes/Edges" schema enforcement.
  + **Error Handling:** Support for multi-lingual rate limit detection (e.g., Chinese error strings).
* **Metric:** Recovery Success Rate and Parsing Accuracy.

**Test Results:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Test Case** | **Status** | **Duration** |
| 1. Retry Logic | Calculate exponential backoff correctly | PASSED | 0.12ms |
|  | Respect max delay cap | PASSED | 0.01ms |
|  | Handle base 1 (no exponential growth) | PASSED | 0.01ms |
|  | Identify network/5xx/429 errors as retryable | PASSED | 0.11ms |
|  | Identify rate limit messages (inc. Chinese) | PASSED | 0.05ms |
|  | Should not retry non-retryable/client errors | PASSED | 0.32ms |
| Circuit Breaker | Response Validation | PASSED | 0.06ms |
| Response Validation | Validate correct response structure | PASSED | 0.09ms |
|  | Validate node/edge data fields | PASSED | 0.06ms |
|  | Require nodes and edges arrays | PASSED | 0.03ms |
|  | Handle nodes with non-object items | PASSED | 0.02ms |
| AI Parsing | Parse valid JSON & markdown code blocks | PASSED | 0.15ms |
|  | Extract JSON from raw text | PASSED | 0.02ms |
|  | Handle invalid JSON gracefully | PASSED | 0.01ms |

Analysis: Resilience tests verify retry boundaries, rate-limit detection, parser safeguards, and circuit-breaker integration behavior. These checks are designed to keep generation stable when upstream providers return partial, delayed, or malformed responses.

**AI Circuit Breaker**

* **Objective:** To verify the finite state machine behavior of the Circuit Breaker, ensuring it accurately transitions between Closed, Open, and Half-Open states based on API failure thresholds.
* **Methodology:** A state-transition validation strategy was used to monitor the internal failure counter and the logic governing request blocking.
  + **States Tested:** Closed (Normal), Open (Blocking), and Half-Open (Recovery Testing).
  + **Key Transitions:** Threshold-triggered opening and success-triggered closing.
  + **Environment:** Circuit Breaker Class unit tests.
* **Metric:** State Transition Accuracy and Execution Blocking Rate.

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Test Case** | **Status** | **Duration** |
| 1. Initial State | Should start in closed | PASSED | 0.07ms |
|  | Should allow failures | PASSED | 0.01ms |
|  | Should count failure on success | PASSED | 0.24ms |
|  | Reset failure count on threshold | PASSED | — |
| 2. Closed State | Should allow failures | PASSED | 0.01ms |
|  | Should block execution when circuit is open | PASSED | 0.04ms |
|  | Should transition the timer after reset timeout | PASSED | 0.08ms |
| 3. Open/Half-Open | Half-Open State validation | PASSED | 0.07ms |
|  | Should track execution when circuit is open | PASSED | 0.1ms |
|  | Transition to closed after successful call | PASSED | 0.07ms |

#### Analysis

Circuit-breaker tests verify provider-isolated state transitions (closed, open, half-open), reset timing, and safe recovery after successful probes. This mechanism limits repeated failing calls and protects overall request latency under provider incidents.

#### Infrastructure Schema Validation

#### Objective

To ensure that all architectural components (Functions, Databases, AI nodes, etc.) adhere to strict structural rules and data types before being processed by the system.

#### Methodology

A boundary-value analysis strategy was used to test both "happy path" validations and intentional schema violations (edge cases).

#### Scope

17 validation points covering 7 distinct infrastructure types.

#### Key Constraints

Validation of numerical ranges (memory, temperature, replicas) and categorical enums (runtime, storage class, platform).

#### Environment

TypeScript-based test suite using Valibot-backed schema contracts and node shape validation (node-schemas.test.ts).

#### Metric

Schema Integrity and Type-Safety Coverage.

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | Test Case | Status | Duration |
| 1. FunctionSchema | Should validate a valid function node | PASSED | 0.19ms |
|  | Should reject invalid runtime / negative memory | PASSED | 0.13ms |
| 2. DatabaseSchema | Should validate a valid database node | PASSED | 0.02ms |
|  | Should reject non-positive storage | PASSED | 0.01ms |
| 3. QueueSchema | Should validate a valid queue node | PASSED | 0.10ms |
|  | Should reject negative visibility timeout | PASSED | 0.02ms |
| 4. ServiceSchema | Should validate a valid service node | PASSED | 0.04ms |
|  | Should reject zero replicas | PASSED | 0.06ms |
| 5. StorageSchema | Should validate a valid storage node | PASSED | 0.02ms |
|  | Should reject invalid storage class | PASSED | 0.03ms |
| 6. AISchema | Should validate a valid AI node | PASSED | 0.02ms |
|  | Should reject temperature out of range | PASSED | 0.07ms |
| 7. ClientSchema | Should validate a valid client node | PASSED | 0.02ms |
|  | Should reject invalid platform | PASSED | 0.03ms |
| 8. Mapping | Should have all required node types | PASSED | 0.03ms |
|  | Should return correct schema for each type | PASSED | 0.04ms |
|  | Should return BaseSchema for unknown type | PASSED | 0.03ms |

#### Analysis

Schema tests confirm strict validation on infrastructure node constraints and ensure robust fallback behavior for unknown node types. This protects downstream rendering and persistence layers from malformed graph payloads.

#### Prompt Engineering & Intent Classification

#### Objective

To verify the system's ability to accurately parse user intent, detect infrastructure architecture types, and build optimized system prompts for LLM consumption.

#### Methodology

A classification-accuracy strategy was used to test the NLP engine against a variety of structural and linguistic patterns.

#### Detection Scope

Includes Web, Mobile, IoT, Blockchain, Serverless, and Monolithic architecture types.

#### Validation Logic

Heuristic-based detection using keyword matching, length analysis, and confidence scoring.

#### Environment

TypeScript-based test runner (prompt-engineering.test.ts).

#### Metric

Classification Accuracy and Validation Rigor.

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | Coverage Summary | Status | Duration |
| 1. Architecture Detection | Validated detection for 10+ types (Web-app, AI/ML, Microservices, etc.). | PASSED | 1.4ms |
|  | Handles mixed architectures and provides follow-up questions. | PASSED | 0.11ms |
|  | Confirms case insensitivity and confidence score calculation. | PASSED | 0.06ms |
| 2. Complexity Detection | Accurately categorizes Simple, Medium, and Complex requests. | PASSED | 0.12ms |
|  | Prioritizes explicit keywords over prompt length for detection. | PASSED | 0.02ms |
| 3. Prompt Validation | Rejects too short, gibberish, or highly repetitive patterns. | PASSED | 0.43ms |
|  | Warns about greeting-only prompts but allows them with requirements. | PASSED | 0.04ms |
| 4. System Prompt Building | Ensures architecture type and mode constraints are included. | PASSED | 0.24ms |
|  | Validates inclusion of complexity guidelines and framework rules. | PASSED | 0.09ms |

#### Analysis

Prompt engineering tests cover architecture detection, complexity classification, prompt validation, and system-prompt composition. These tests improve consistency of generated outputs across diverse prompt styles and ambiguity levels.

#### API Infrastructure & Data Integrity

#### Objective

To verify that the API's entry points (Project Creation and Generation Requests) enforce strict data contracts and sanitize input before it reaches the core logic.

#### Methodology

A multi-layered validation strategy was used to test both specific business objects and primitive data types.

#### Schema Depth

Includes validation for complex objects (CreateProject), standard identifiers (UUIDs), and contact information (Emails).

#### Resilience Strategy

Specifically tested against "poisoned" inputs, such as deeply nested invalid data and incorrect structural types (e.g., providing an array when an object is required).

#### Environment

API and schema contract tests are implemented in Vitest suites, with deep edge-case coverage for malformed payloads and invalid primitive types.

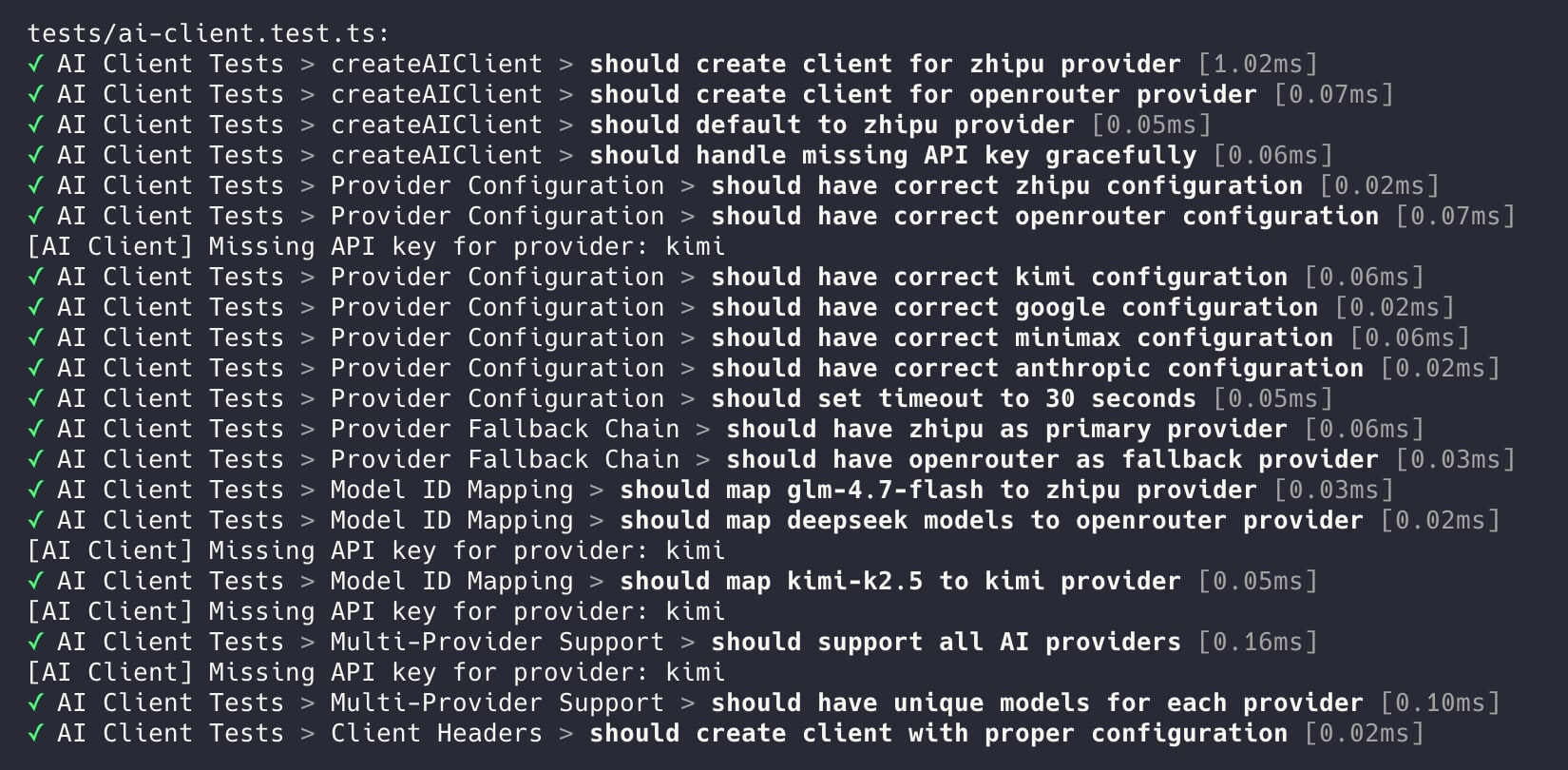
#### Metric

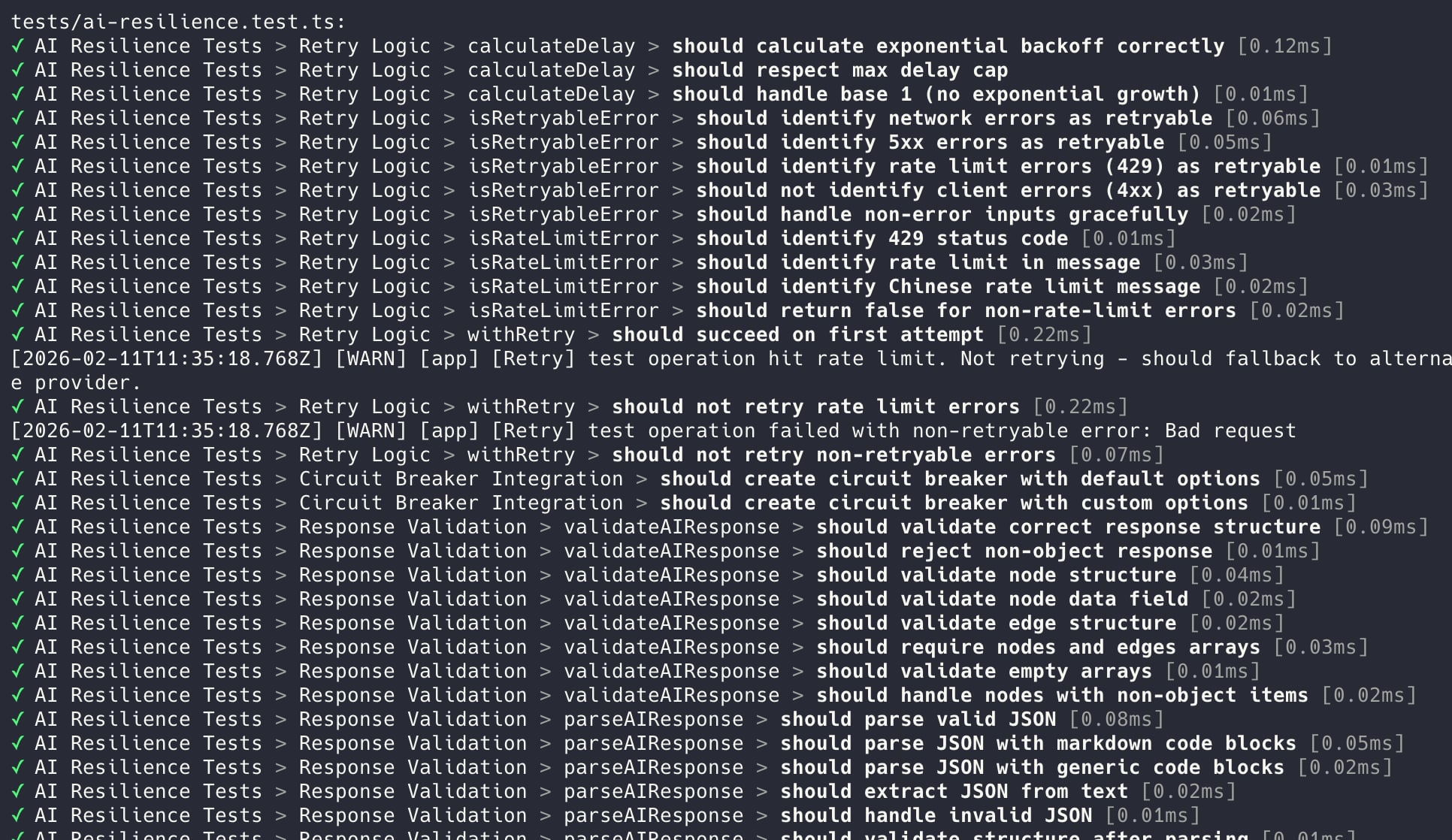
Data Sanitization Efficiency and Schema Enforcement Accuracy.

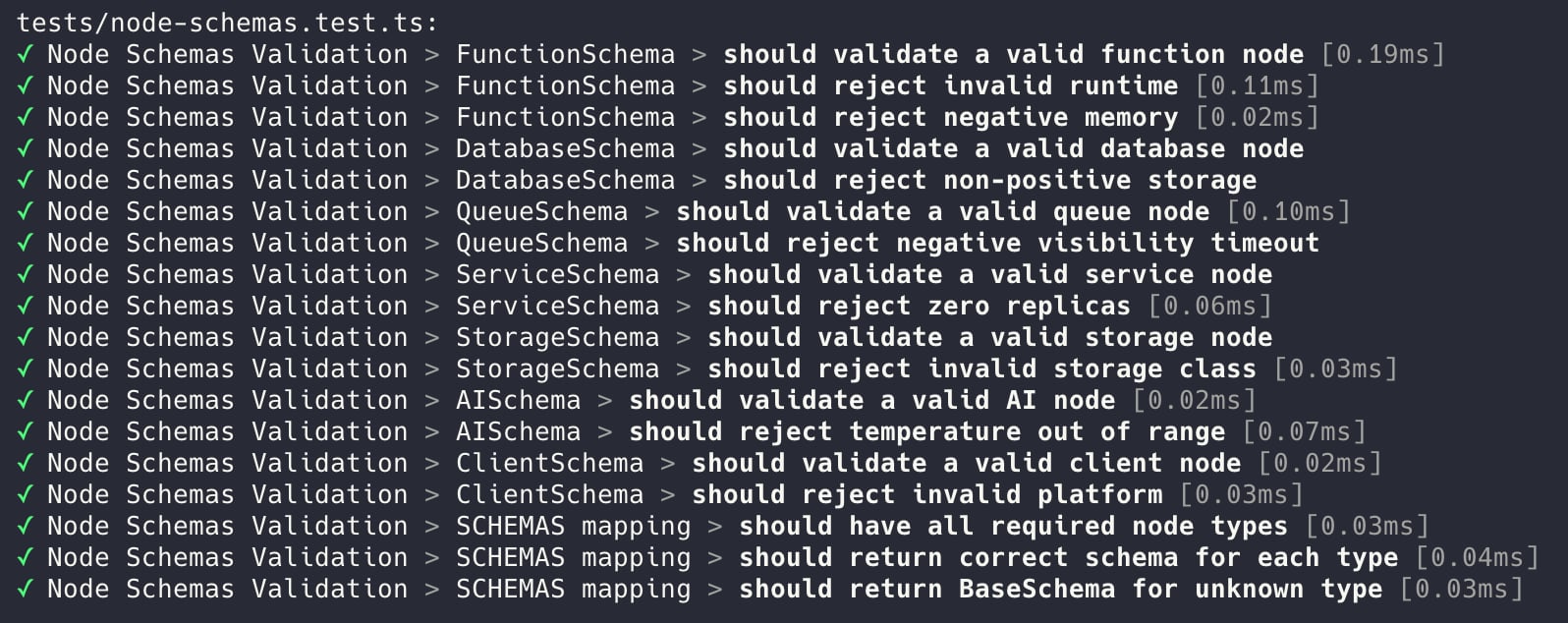
|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | Test Case | Status | Duration |
| 1. CreateProject Schema | Should validate minimal project creation | PASSED | 0.03ms |
|  | Should catch empty project name | PASSED | 0.03ms |
|  | Should catch project name exceeding max length | PASSED | 0.03ms |
|  | Should validate complete project with all optional fields | PASSED | 0.02ms |
| 2. GenerateRequest Schema | Should validate valid generation request | PASSED | 0.23ms |
|  | Should catch empty prompt | PASSED | 0.10ms |
|  | Should catch invalid mode value | PASSED | 0.09ms |
|  | Should accept request with only required prompt field | PASSED | 0.03ms |
| 3. Data Type Schemas | Should validate valid UUID / catch invalid format | PASSED | ~0.09ms |
|  | Should validate valid Email / catch invalid format | PASSED | ~0.08ms |
|  | Should validate non-empty string / catch whitespace-only | PASSED | ~0.04ms |
| 4. Edge Cases | Should handle null/undefined values gracefully | PASSED | ~0.04ms |
|  | Should handle wrong data types (array instead of object) | PASSED | 0.02ms |
|  | Should validate deeply nested invalid data | PASSED | 0.08ms |

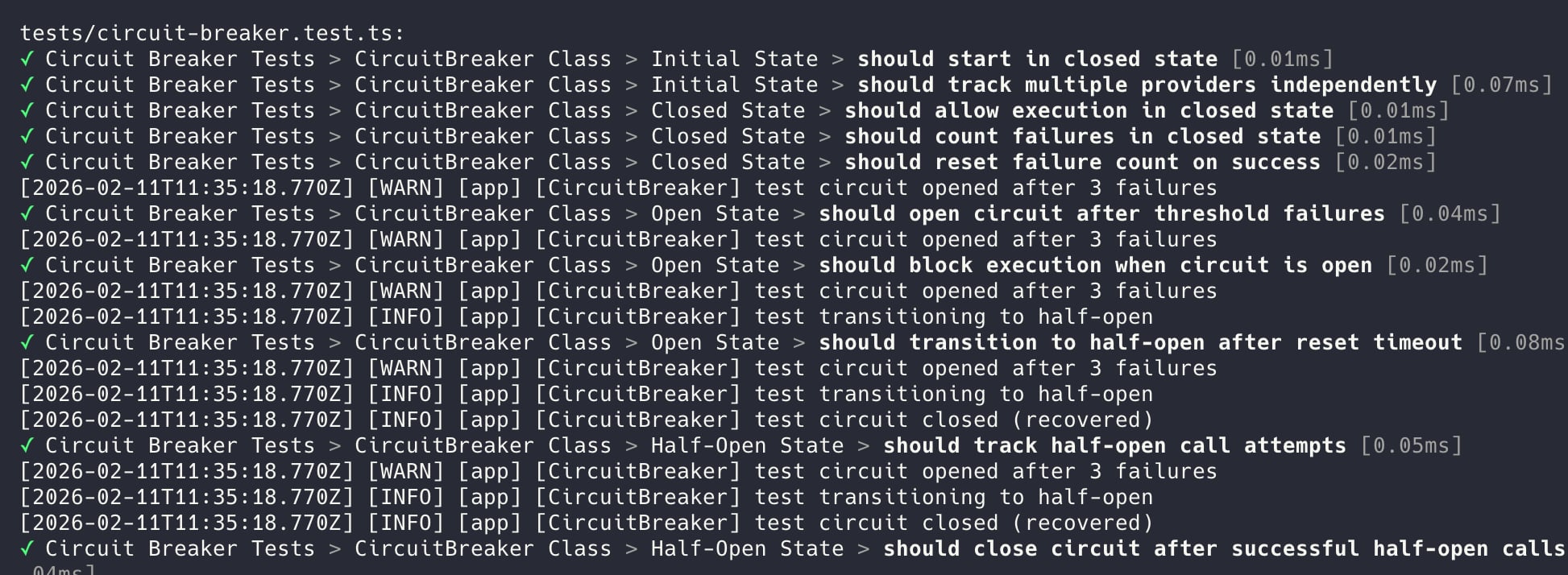
#### Analysis

API schema tests validate input contracts for project creation and generation requests, including UUID/email/string guards and nested-structure error handling. This hardens entry points before data reaches generation and persistence logic.

****





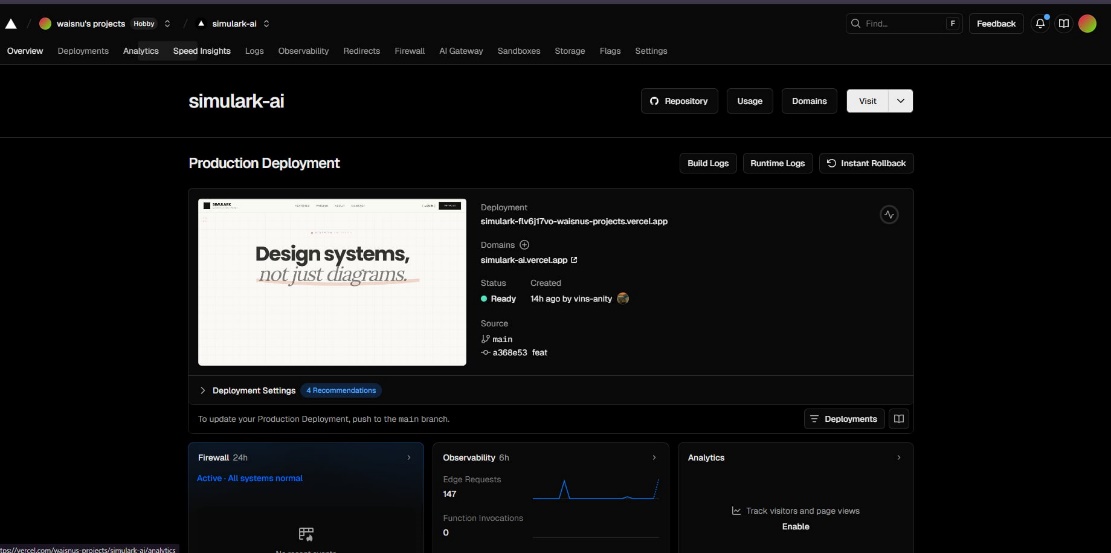


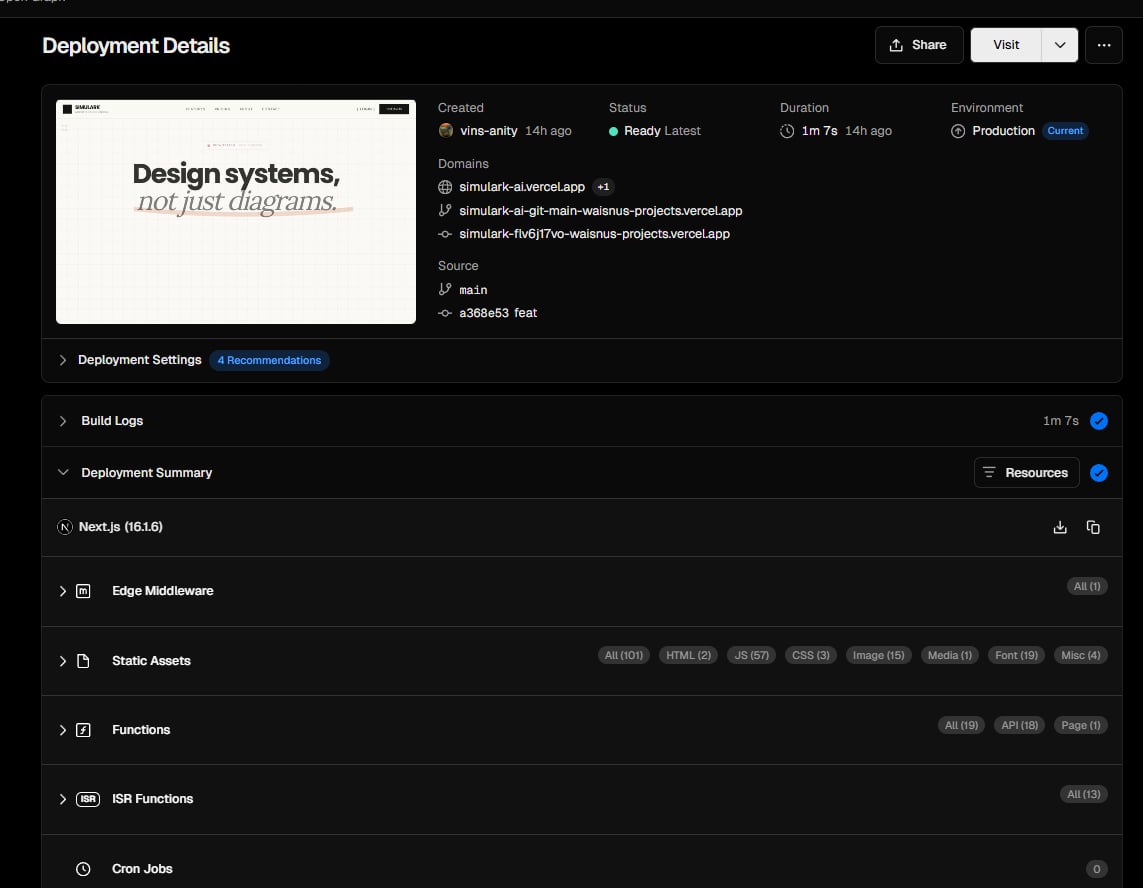
# 

# 

# Deployment and Integration

#### Deployment Strategy

****

****

The Simulark project is deployment-ready for Vercel with Next.js 16 on Bun runtime. Key deployment characteristics:

**Infrastructure:**

* **Database**: Supabase (PostgreSQL) with Row-Level Security
* Caching: Upstash Redis integration plus response-caching utilities for repeated generation paths.
* AI Services: NVIDIA-routed models, ZhipuAI, OpenRouter, and Kimi-compatible endpoints with fallback controls.

**Security Strategy:**

* CSP headers with service allowlists
* Tiered rate limiting in app and API layers (burst controls plus daily generation quotas by subscription tier)
* Comprehensive security headers (HSTS, X-Frame-Options, etc.)

**Deployment Flow:**

1. bun run build -> Next.js production build (Bun runtime)
2. vercel --prod -> Deploy to Vercel
3. Health check at /api/health monitors Supabase + Redis status

**Rollback:**

* vercel --prod --yes for immediate revert
* Supabase point-in-time recovery for database rollback

# Documentation and Presentation

**Summary of Achievements:**

* Interactive Architecture Canvas with XYFlow and semantic nodes
* AI-Powered Architecture Generation with multi-provider fallback
* Visual Simulation & Resilience Testing (Chaos Mode, congestion detection)
* Context Bridge for IDE integration (Cursor/Windsurf)
* Type-safe full stack with Valibot, TypeScript strict mode, and Biome formatting/linting
* Production-Ready Infrastructure (Supabase, Redis, security headers)

**Lessons Learned:**

* AI output validation complexity with multiple providers
* State synchronization in canvas with React Flow
* Multi-provider AI fallback strategy design
* Server Actions vs API Routes trade-offs
* Bundle size management with heavy UI libraries

**Future Work:**

* Complete billing/payment integration and automate subscription lifecycle jobs
* Collaborative editing with real-time sync
* Terraform/cloud deployment export
* Mobile responsiveness improvements
* Template marketplace, custom node SDK, and expanded context bridge capabilities

# References

* **Core Framework & Runtime:** (Next.js 16, React 19, Bun, TypeScript)
* Frontend Libraries: (React, Tailwind CSS v4, Shadcn/UI, Radix UI, XYFlow, Zustand, React Hook Form)
* **Backend & API:** (Next.js API Routes, Server Actions, Valibot)
* **Database & Authentication:** (Supabase, PostgreSQL, Supabase Auth)
* **AI Integration:** (OpenAI SDK, ZhipuAI, OpenRouter, Kimi)
* **State Management & Caching:** (Zustand, Redis)
* **Development Tools:** (Biome, Vite, Vitest, TypeScript Strict Mode)
* Infrastructure & Deployment: (Supabase, Vercel, Upstash Redis)
* **UI & Styling:** (Tailwind CSS v4, CSS variables, clsx, tailwind-merge)
* Key dependencies: (openai, ai/@ai-sdk, @supabase/ssr, @t3-oss/env-nextjs, valibot, jspdf, jszip)

# Appendices

Appendix A - Test Evidence Index: Detailed test-case tables for AI client, resilience, circuit breaker, schemas, and API validation.

Appendix B - Architecture and UI Evidence: Screenshots and render outputs of canvas workflows, generated diagrams, and editor states.

Appendix C - Deployment and Environment Notes: Runtime assumptions, environment variables, provider configuration, and rollout references.

Appendix D - API and Data Contract Notes: Route inventory, schema references, and validation constraints used in implementation.