**Project Report**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **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 - 3
2. Background and Problem Statement - 7
3. Project Proposal and Planning - 8
4. System Design and Architecture - 10
5. Data Preparation and Processing - 14
6. Application Development and Testing - 16
7. Deployment and Integration - 38
8. Documentation and Conclusion - 40
9. References - 41

**Document Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Effective Date of release** | **Details** | **Author** |
| 1.0 | 15 Nov 2025 | Initial Creation |  |

# Project Overview

### Introduction

In the contemporary software engineering landscape, there is a widening gap between the speed of code generation and the maturity of architectural design. While AI-driven tools like GitHub Copilot have significantly accelerated the coding process, they often operate in a vacuum, lacking the "big picture" perspective required for complex systems. This discrepancy leads to a phenomenon known as **"Context Loss,"** where the high-level design of a system becomes disconnected from the actual implementation.

Traditional architectural tools, such as Lucidchart, provide static visualizations where components are merely shapes on a screen. For instance, a rectangle labeled "Queue" in a standard diagram lacks any inherent logic or functional awareness. As systems transition from monolithic structures to intricate microservices, the mental models required to manage them become too large for individual developers to maintain, leading to an estimated **30% loss in development time** due to context switching and architectural misalignment.

**Simulark** is introduced as a "Generative UI" platform designed to transform natural language prompts into active, semantic architecture diagrams. Unlike its static predecessors, Simulark treats nodes as **semantic entities**; a "Queue" node within this platform understands its role in generating asynchronous traffic.This living blueprint approach offers several key innovations such as Semantic Understanding that enables real-time simulations of system resilience, AI-Ready Context that Generates machine-readable JSON exports to provide AI coding agents with the necessary architectural context, and Automated translation that utilizes a Multi-Agent AI system to translate complex text descriptions into structured, valid system graphs.

## 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 Generative Architecture Engine using a Multi-Agent AI System (Orchestrator Pattern) that translates text into JSON graphs.
2. To develop an **Interactive Canvas** using xyflow (React Flow) that visualizes 100+ nodes with high performance.
3. To create a **Schema Validation Layer** using Valibot to strictly enforce architectural correctness (e.g., ensuring Queues connect to Consumers).
4. To implement Rate Limiting Protection using PostgreSQL RPC functions to manage API costs effectively.

## 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 system employs a multi-agent orchestration layer that integrates **ZhipuAI (GLM-4)** to handle complex architectural reasoning and **Mistral Small** (via OpenRouter) to generate the strict, machine-readable JSON schemas required for the diagram nodes.

### 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:** The core generative functionality of the platform is entirely dependent on active API connections to **OpenRouter** and **ZhipuAI**. While cached diagrams may be viewable, the "Generative Engine" cannot function in an offline environment.

### 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 centered on the integration of the generative engine. The core logic was centralized in actions/ai-orchestrator.ts, facilitating the multi-agent communication between **ZhipuAI** and the JSON generation layer. To enhance user experience, a "Thinking" UI was developed to provide real-time feedback during the LLM’s reasoning process.
  + **Sprint 3:** The final sprint focused on adding semantic depth and production readiness. This included the development of the **traffic animation loop** to visualize data flow between nodes. Finally, the application was optimized for performance and global availability by deploying the entire stack to the **Vercel Edge Network**.

# Background and Problem Statement

### Context and motivation

The motivation for this project stems from the increasing complexity of distributed systems. As systems move from Monoliths to Microservices, the mental model of the system becomes too large for a single developer to hold. Anecdotal evidence suggests that 30% of development time is lost due to "Context Switching" and misalignment between the architectural diagram (often out of date) and the codebase.

### 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. **Lack of Remote Monitoring:** Software architects and developers struggle to manually translate complex natural language requirements into structured, machine-readable blueprints, leading to human error in initial design phases.
4. **Non-Deterministic Reasoning:** Without a structured "Aggregator" or planning layer, generative AI frequently "hallucinates" incompatible architectural connections, such as attempting to link storage buckets directly to end-users without necessary distribution layers.

### Assumptions

### The development and evaluation of Simulark are predicated 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:** Given that the "Generative Engine" relies on external multi-agent API orchestration for reasoning and schema generation, a stable internet connection is assumed to be available. Offline functionality is expected to be limited to the viewing of locally cached architectural diagrams.

# Project Proposal and Planning

Timeline

|  |  |  |
| --- | --- | --- |
| **Phase** | **Dates** | **Deliverables** |
| **Week 1: The Core** | **Jan 3 – Jan 9** | **Setup & Database**   * Setup Next.js 16 + Supabase * Create Database (Users, Queue Tables). * **Goal:** A functional web platform where users can authenticate and persist a basic workspace layout, serving as the "canvas" for future AI-generated architectural nodes. |
| **Week 2: The Brain** | **Jan 10 – Jan 16** | **AI Implementation**   * **Multi-Agent Integration:** Develop the core AI pipeline by integrating **ZhipuAI (GLM-4)** for high-level architectural reasoning and **Mistral Small** (via OpenRouter) for structured data output. * **Prompt Engineering & Validation:** Author specialized system prompts to transform natural language into valid JSON graphs, using **Valibot** to enforce connection rules and prevent architectural hallucinations. * **Logic Implementation:** Write the actions/ai-orchestrator.ts logic to manage the handover between the "Aggregator" agent (planning) and the "Generator" agent (execution). * **Goal:** A "Thinking" UI where a user inputs a prompt like *"Design a serverless API"* and the system provides real-time status updates (e.g., "[Aggregator] Planning... [Generator] Mapping Nodes") before rendering the final diagram. |
| **Week 3: The Polish** | **Jan 17 – Jan 22** | **UI Polish & Connection**   * **Interface Refinement**: Enhance the React frontend using **TailwindCSS v4** and **Framer Motion** to ensure a high-fidelity user experience. This includes styling the node-based canvas, custom control panels, and the "Thinking" logs to maintain professional aesthetic standards. * **Full-Stack Synchronization**: Establish seamless communication between the **Next.js** frontend and the **Supabase/Edge Function** backend. This ensures that AI-generated architectural graphs are correctly parsed, validated via **Valibot**, and persisted in the database without latency issues. * **Goal**: A fully integrated, production-ready demonstration capable of showcasing the end-to-end "text-to-architecture" workflow in a live environment. |
| **Final Prep** | **Jan 23 – Jan 24** | **Documentation & Defense**   * Write the paper. * Create the slide deck. * **Goal:** Submission. |

**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 (ZhipuAI & OpenRouter)**: As the platform relies on these APIs for reasoning and JSON generation, these providers are technical stakeholders whose service reliability and model updates directly impact Simulark’s performance.

**3. Tertiary Stakeholders**

* **Academic Evaluators**: In the context of this capstone project, the faculty and thesis advisors are stakeholders who evaluate the methodology, the 92% graph generation success rate, and the technical implementation of the multi-agent system.

**Tools and Technology**

|  |  |  |
| --- | --- | --- |
| **Category** | **Technology** | **Purpose** |
| **Frontend** | Next.js 16 (React Server Components), TailwindCSS v4, Zustand (State Management). | User interface (App Router). See package.json scripts and app for pages and API routes. |
| **Backend** | Supabase (PostgreSQL 16), Edge Functions. | Database, auth, and server-side helpers |
| **AI** | OpenRouter API, ZhipuAI API. | Will provide the graphs/response to user request |
| **Tools** | Cursor AI Editor, v0.dev (for initial UI scaffolding). | Developmental tools for more improved development |

# 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 layer employs a sophisticated multi-agent pipeline to generate architectures that are not just visually appealing but technically sound. The system follows a reasoning-first approach, leveraging models with “Deep Thinking” capabilities such as GLM-4.7 Flash from ZhipuAI. The pipeline begins with an Aggregator Agent, which uses Upstage Solar Pro to analyze architectural constraints such as scalability, consistency, and availability before generating any visual elements. This agent produces a high-level architectural plan that serves as the foundation for the next stage. The Generator Agent then transforms that plan into a strict JSON graph structure using Mistral Small 3.1, which minimizes hallucinations by enforcing schema constraints. This two-agent approach separates the planning reasoning from the technical implementation, resulting in more robust and reliable architecture generation.

**Multi-Provider AI Strategy**

Simulark implements a robust fallback strategy to ensure high availability across different AI providers. The primary provider is ZhipuAI using their GLM-4.7 Flash model, which offers excellent performance for architectural reasoning tasks. When the primary provider fails or rate limits are exceeded, the system seamlessly falls back to OpenRouter, which provides access to multiple models including Arcee AI Trinity, Mistral Small, and Google Gemma. This hybrid approach ensures that users can always generate architectures regardless of individual provider issues. The AI client abstraction layer handles provider switching transparently, with each provider configured with specific models, base URLs, and reasoning parameters optimized for their respective capabilities.

**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 follows Next.js API route conventions to expose several key services. The generation endpoint implements Server-Sent Events to stream the AI’s thought process directly to the UI, enhancing transparency and user engagement. This streaming approach allows users to see the reasoning process unfold in real-time, making the AI’s decision-making more interpretable. Other endpoints handle project CRUD operations, chat history management, and the context bridge functionality. The context bridge is particularly important for the modern AI-assisted workflow, exposing secure read-only JSON endpoints that represent the current architectural state and automatically generating configuration files for IDEs like Cursor and Windsurf.

**Database and Authentication Layer**

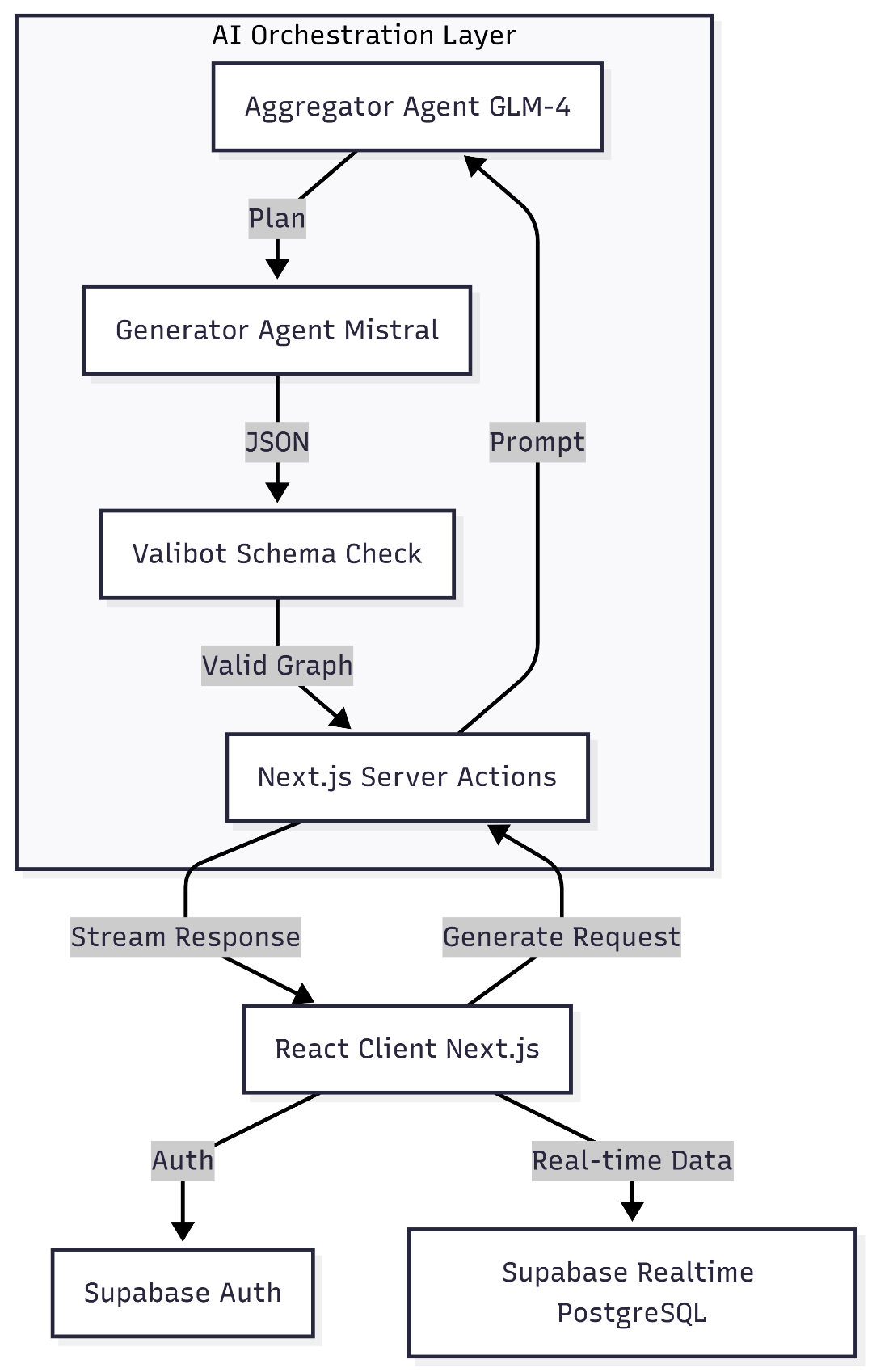
Supabase handles authentication, persistent storage, and complex data relationships through PostgreSQL. The database schema includes tables for users, projects, AI generations, chats, and messages, all protected by Row-Level Security policies that ensure strong data isolation between users. A custom rate limiting system uses PostgreSQL RPC functions to manage API usage limits based on user subscription tiers, with free users limited to 10 generations per day, pro users to 50, and enterprise users to 1000. This usage tracking is integrated directly into the generation workflow through the orchestrator action, which checks and increments usage before processing any generation request.

**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**

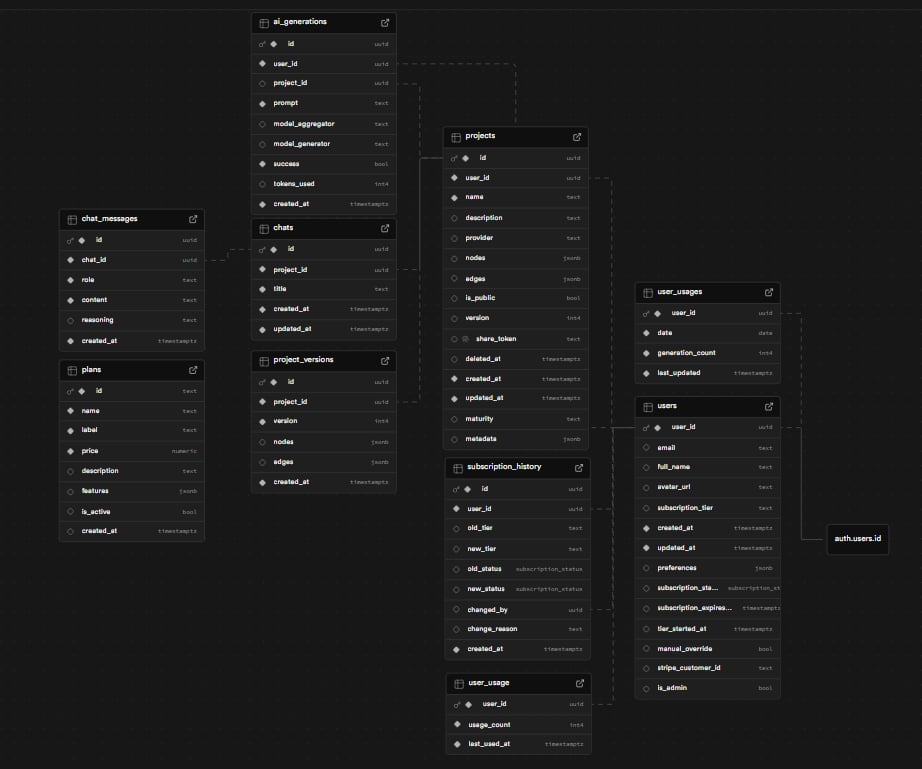
The system supports three project archetypes that influence how architectures are generated. The startup archetype, represented by a "Lean Startup CTO" persona, prefers managed services and high-level abstractions like Next.js, Supabase, and Vercel, avoiding complex infrastructure like Kubernetes unless critical. The corporate archetype, represented by an "Enterprise Architect" persona, prioritizes robustness, high availability, security, and uses established enterprise patterns including microservices, gRPC, and event-driven architecture with Kafka. The default archetype balances modern fullstack best practices with reliable tools. These archetypes automatically adjust the technology selection based on the user's natural language prompt, choosing appropriate frontend frameworks, backend ecosystems, databases, and infrastructure based on the specific requirements and context of the request.



## Database Design (Data Model)

The system will follow a serverless architecture using Supabase as the BaaS). Front end remains Next.js (App Router) and uses Supabase for real-time data and authentication. AI processing can run as a serverless function (Next.js API route / Edge function) or as an external Python microservice — both integrate with Supabase for data storage and eventing.

#### Entity Relationship Diagram (ERD)



**3.2 Collection Schema**

The conceptual model below defines the relationships between the three core entities: Users, Branches, and Queue Sessions.

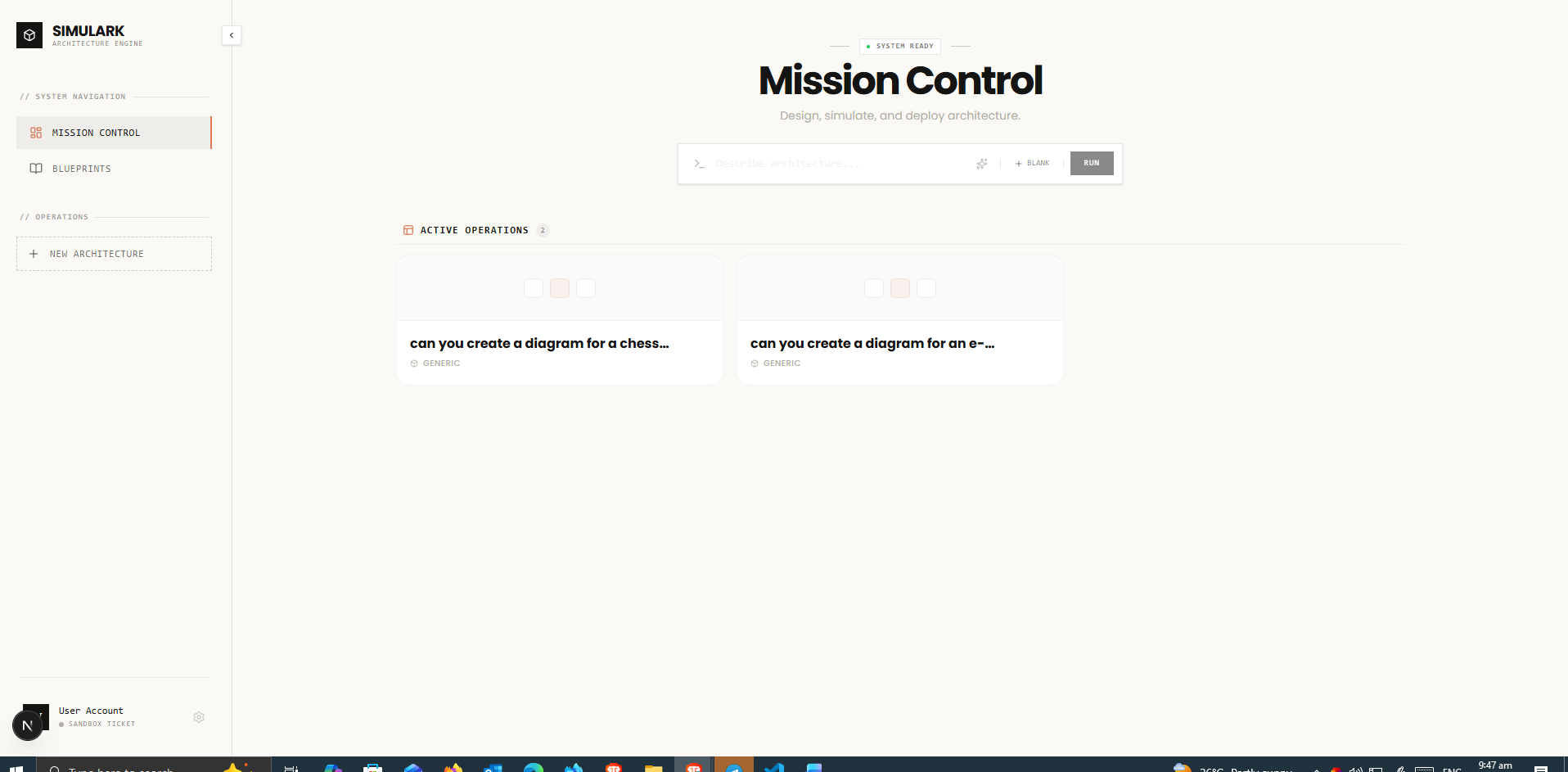
* + users table: Stores the state of every user.
    - id (PK): Unique Identifier.
    - email: Email of the user.
    - subscription\_tier (FK): subscription based on the user’s choice/input
    - preferences: User specific settings
  + projects table: Represents physical locations.
    - id (PK): Unique Identifier
    - name: Name of the project
    - nodes: Stores graph node objects
    - edges: Stores graph edge definitions
    - is\_public: Indicator whether a project is visible to other users
  + ai\_generations: Represents a game session.
    - id (PK): Unique Session ID.
    - prompt: User input/request
    - model\_used: LLM model of choice
    - success: Boolean status
    - tokens\_used: Records how many model tokens an AI call consumed
  + subscription\_tiers
    - `id` (PK): Unique identifier
    - `name`: Tier name
    - `display\_name`: Display name
    - `price\_monthly`: Monthly price
    - `price\_yearly`: Yearly price
    - `features`: Tier features list
    - `limits`: Usage limits
    - `is\_active`: Tier availability
* project\_versions:
  + project\_id: Links back to the main project.
  + version: Integer tracking the iteration.
  + nodes: graph nodes
  + edges: graph edges
* Chats
  + id (PK): Unique identifier
  + project\_id (FK): References project id
  + title: chat session title
* chat\_messages:
  + id` (PK): Unique identifier
  + chat\_id  (FK): References chat id
  + role: Message role
  + content:Message content
  + reasoning: Reasoning content for AI responses
* user\_usages:
  + user\_id (PK): References auth users
  + date: Date
  + generation\_count: Numbers of generations used
  + last\_updated: Last update timestamp
* subscription\_history
  + id (PK): Unique Identifier
  + user\_id (FK): References users.user\_id
  + old\_tier: Previous subscription tier
  + new\_tier: New subscription tier
  + old\_status: Previous status
  + new\_status: Previous status
  + changed\_by: Admin who made the change
  + change\_reason: Reason for the change

## User Interface Design (Frontend)

**4.1 Mission Control Dashboard Prototype**

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

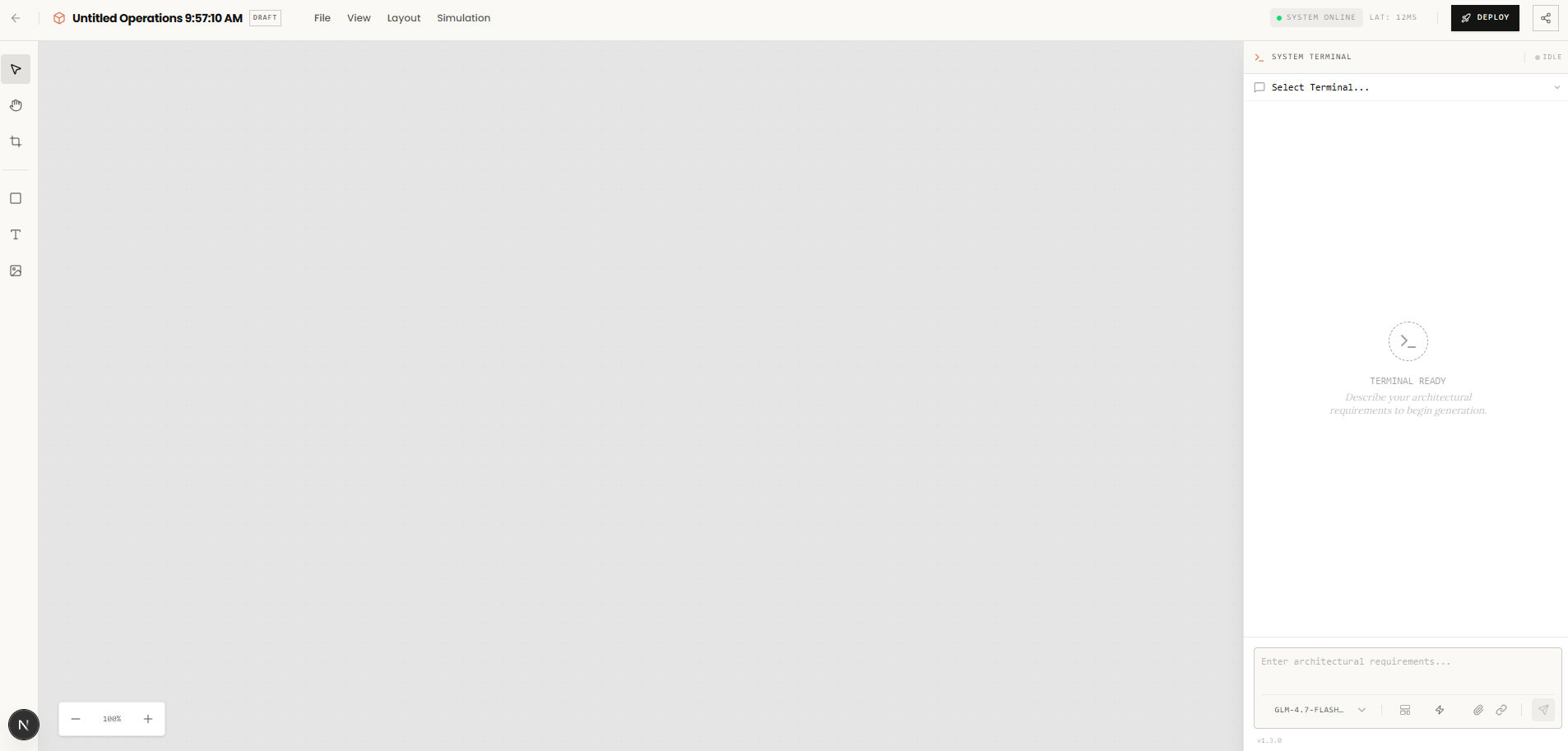
* **System Navigation:** A persistent left-hand sidebar that categorizes the workflow into "System Navigation" (Mission Control, Blueprints) and "Operations," allowing users to pivot between high-level overviews and specific builds.
* **Command Bar:** A central, prominent text input field labeled "Describe 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**: upstage/solar-pro-3:free
* 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

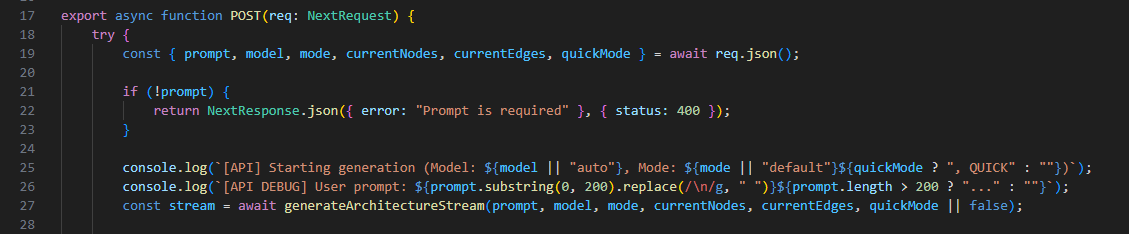
Simulark leverages multiple data sources to deliver intelligent backend architecture design: **AI Provider APIs** (OpenRouter for models like Solar Pro and Mistral Large, and ZhipuAI for GLM-4.7 Flash with "Deep Thinking" capabilities) power the architecture generation engine, enabling semantic analysis and multi-agent orchestration; **Supabase (PostgreSQL)** serves as the primary database for persistent storage of user projects, chat history, and architectural diagrams with Row-Level Security for data isolation; **Tech Ecosystem Database** contains comprehensive technology definitions across 80+ tools and services spanning frontend frameworks (React, Next.js), backend runtimes (Node.js, Go, Python), databases (PostgreSQL, MongoDB, Redis), cloud platforms (AWS, GCP, Azure), serverless functions (Lambda, Cloudflare Workers), and messaging systems (Kafka, RabbitMQ, SQS); and **User Input** via the interactive canvas where natural language requirements and manual node manipulations feed into the AI pipeline to generate and refine architecture diagrams.

**Feature Engineering:**

AI Architecture Generation

Data Structure:

* prompt – User's architecture request
* model – Selected AI model
* mode – default | startup | corporate
* quickMode – Fast generation flag
* currentNodes/currentEdges – Existing architecture context



Multi-Provider Fallback

**Providers:**

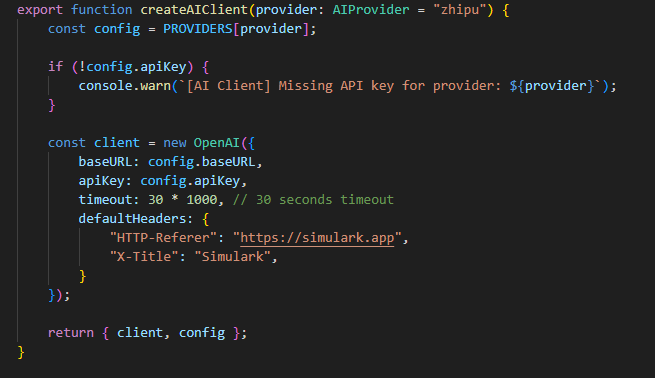
* zhipu – Primary (glm-4.7-flash)
* openrouter – Fallback (arcee-ai/trinity-large-preview:free)



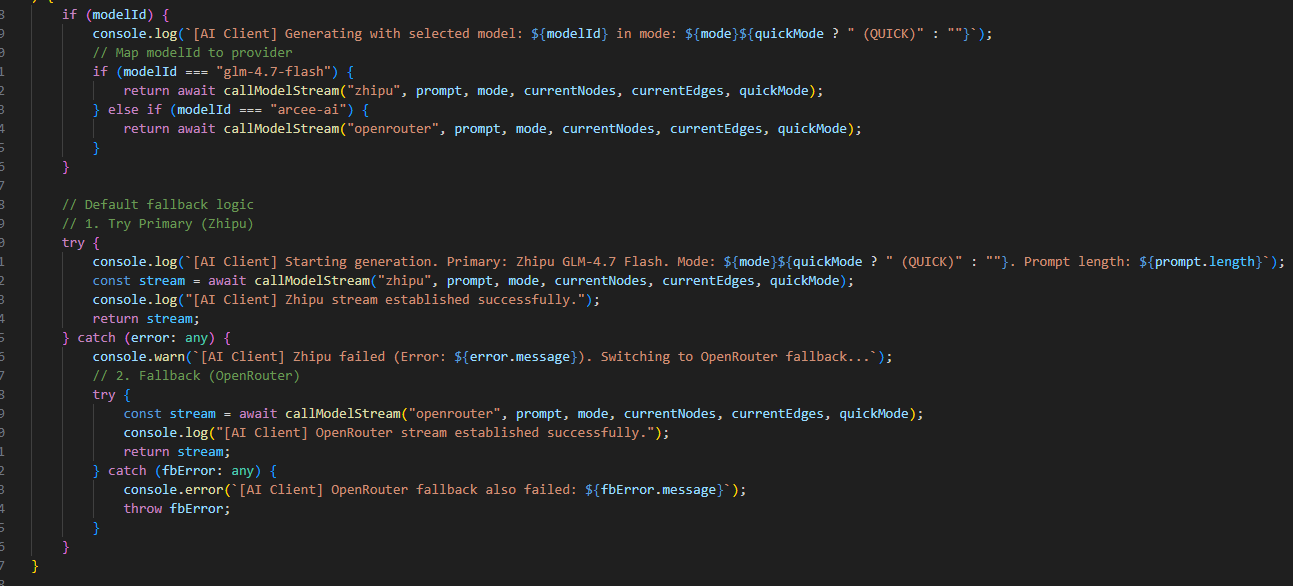
Provider Configuration



Client Creation



Fallback Logic



Architecture Graph Output

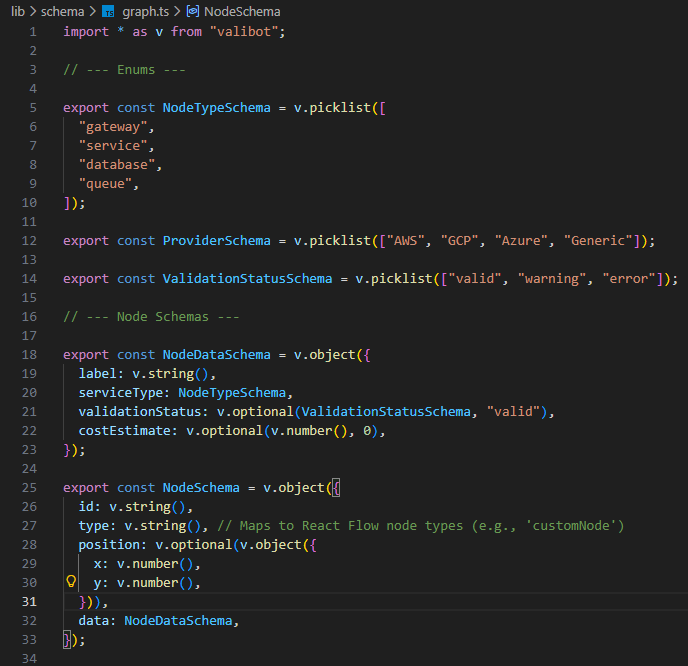
**Nodes:**

* id – Unique identifier
* type – gateway | service | database | queue | ai | frontend | backend
* position – { x, y } coordinates
* data.label – Display name
* data.tech – Tech stack (nextjs, postgres, aws, etc)
* data.description – Component purpose

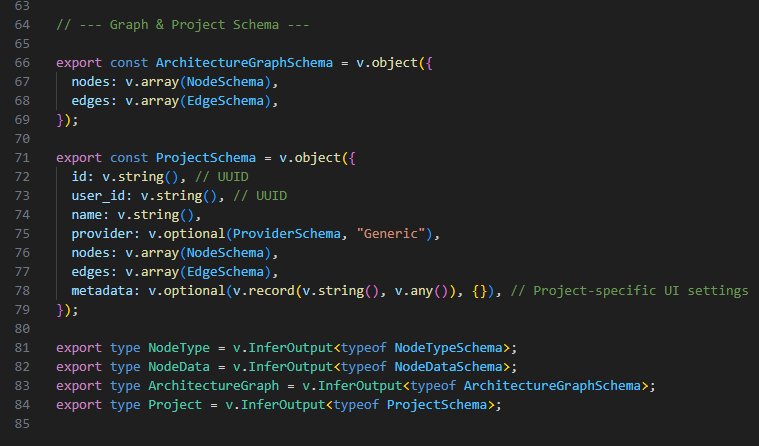
**Edges:**

* id – Unique identifier
* source/target – Node IDs
* data.protocol – http | graphql | websocket | queue | etc

Schema Definitions



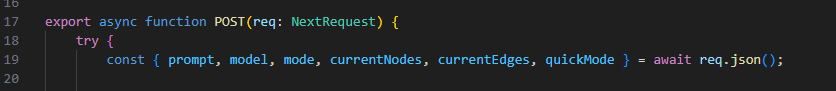




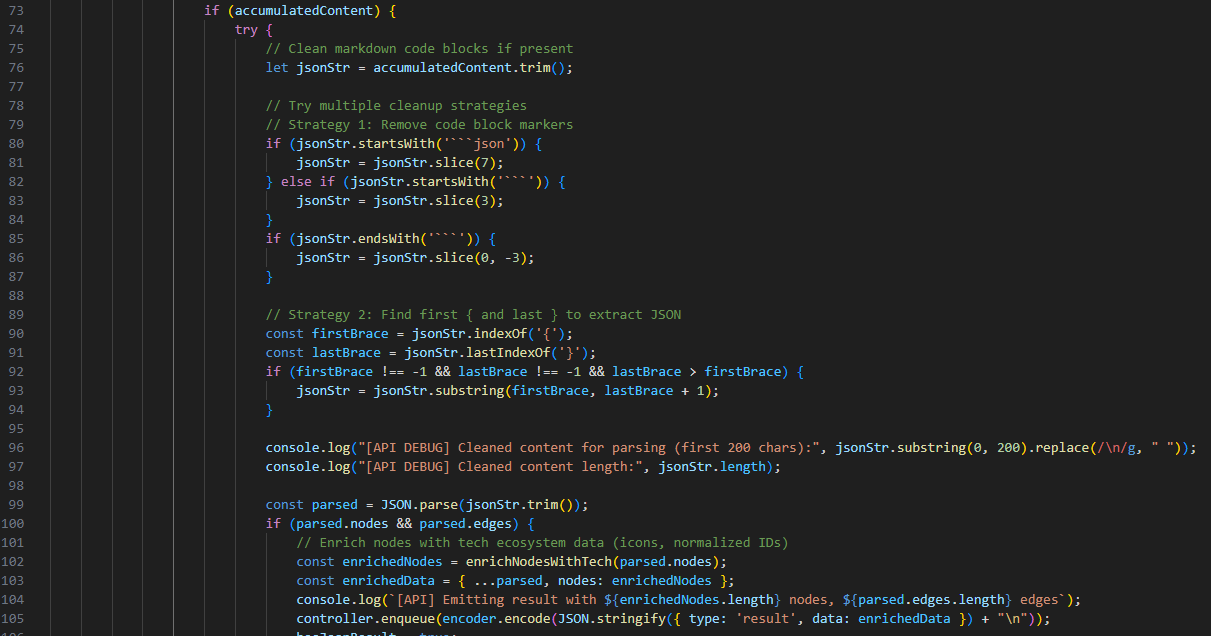
AI Prompt Template



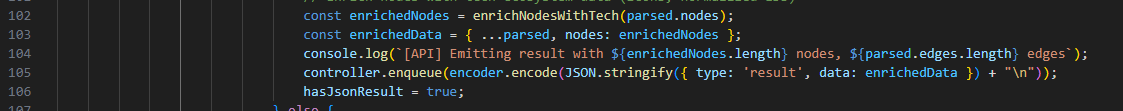
JSON Parsing



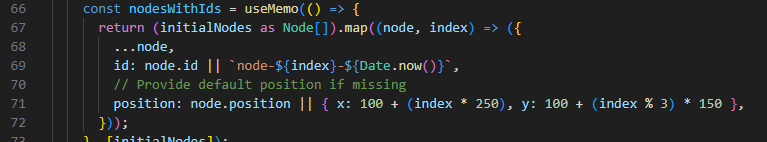




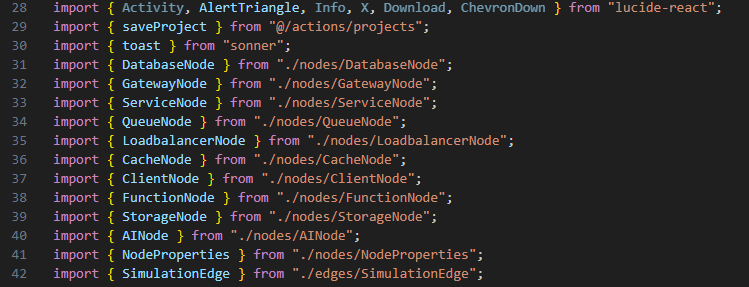




Rendering







Rate Limiting

**Tiers:**

* free – 10 generations/day
* pro – 50 generations/day
* enterprise – 1000 generations/day

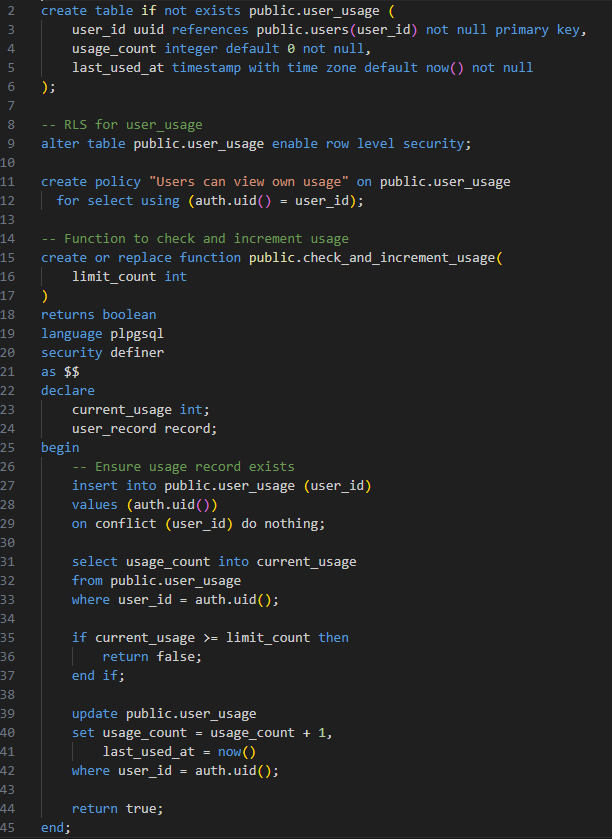
Rate limiting check



Subscription plans



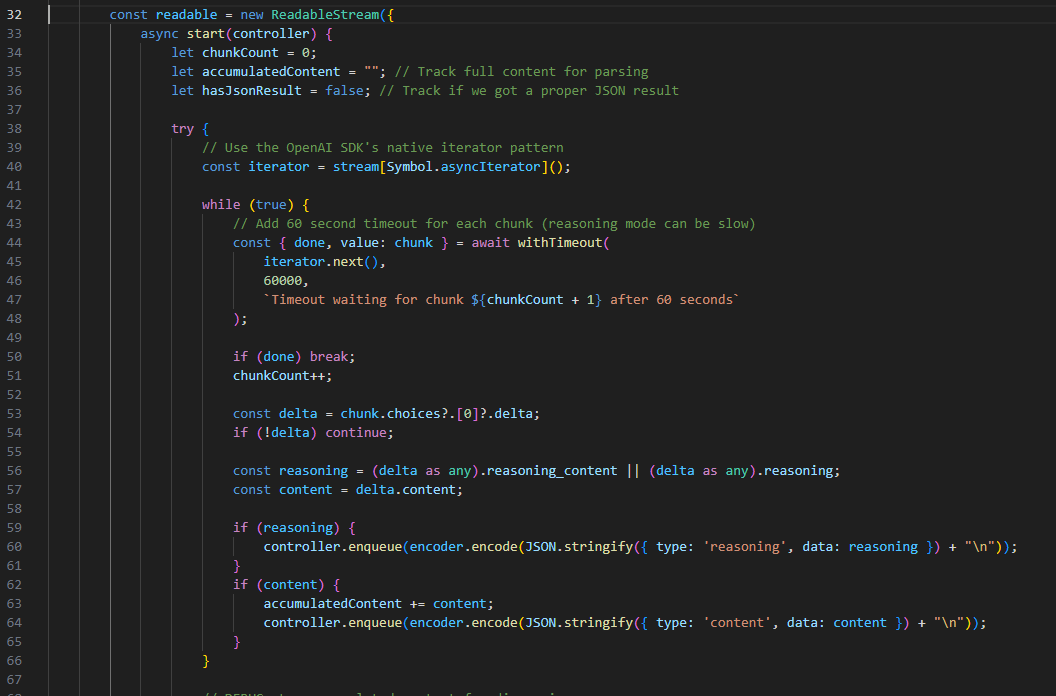
Database Schema

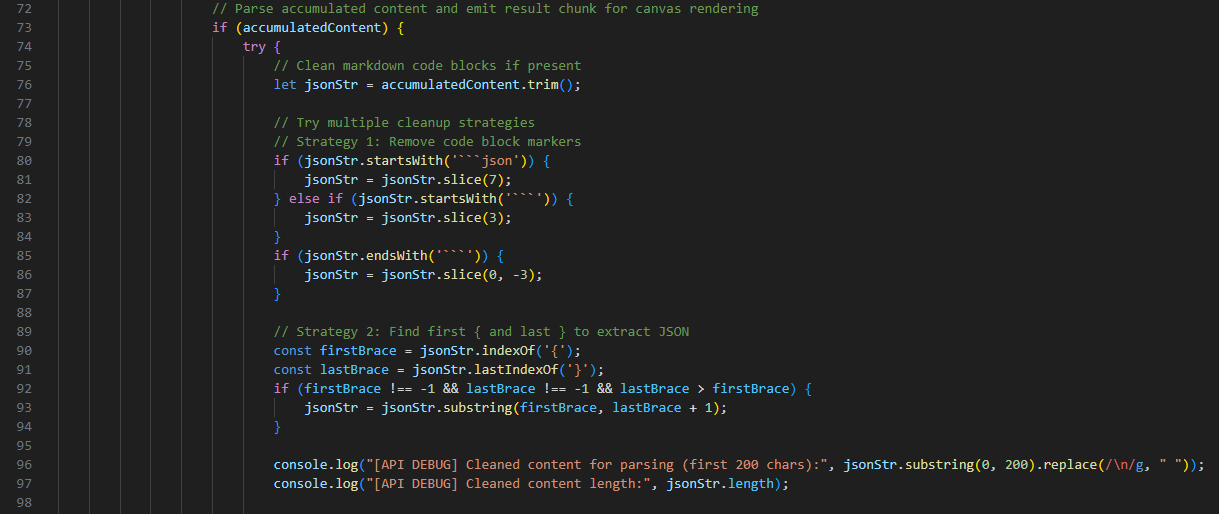


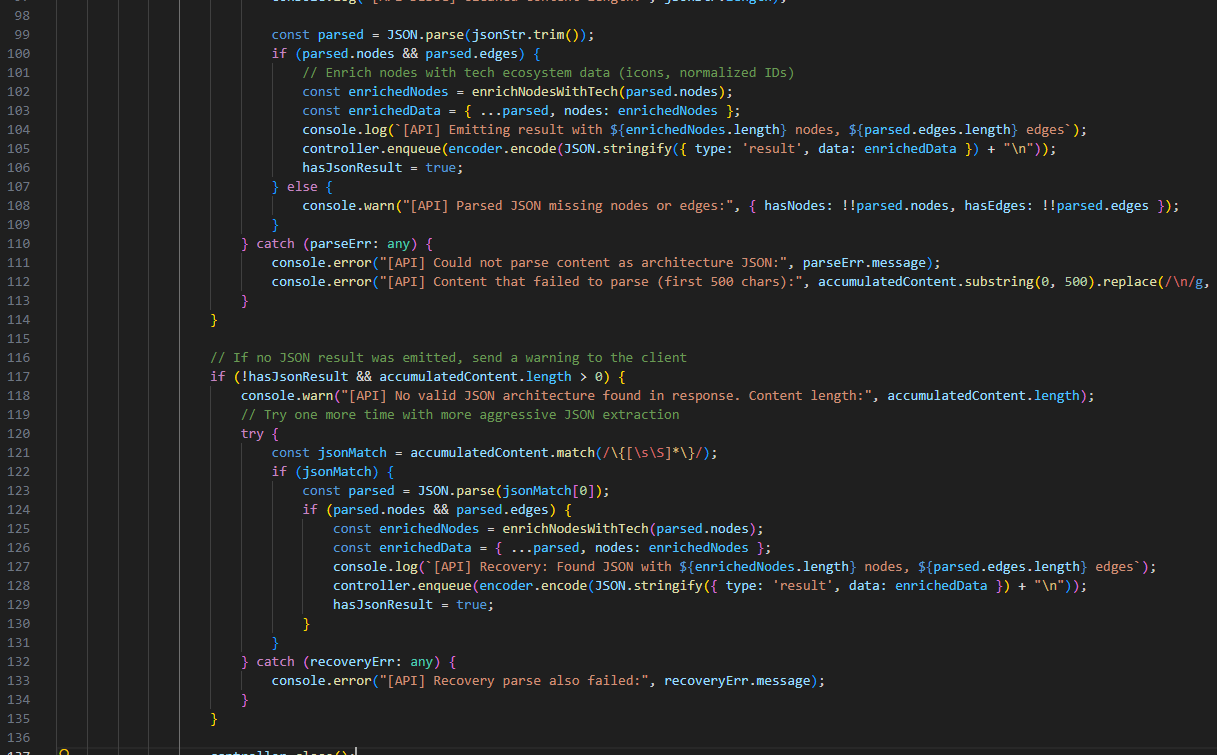
Streaming Response

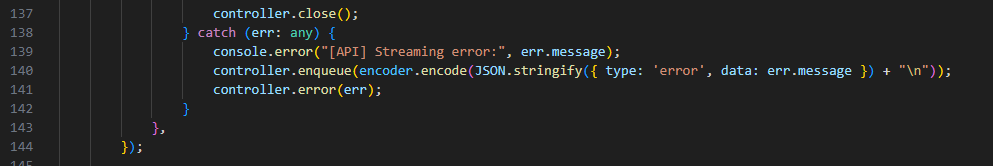
**Types:**

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





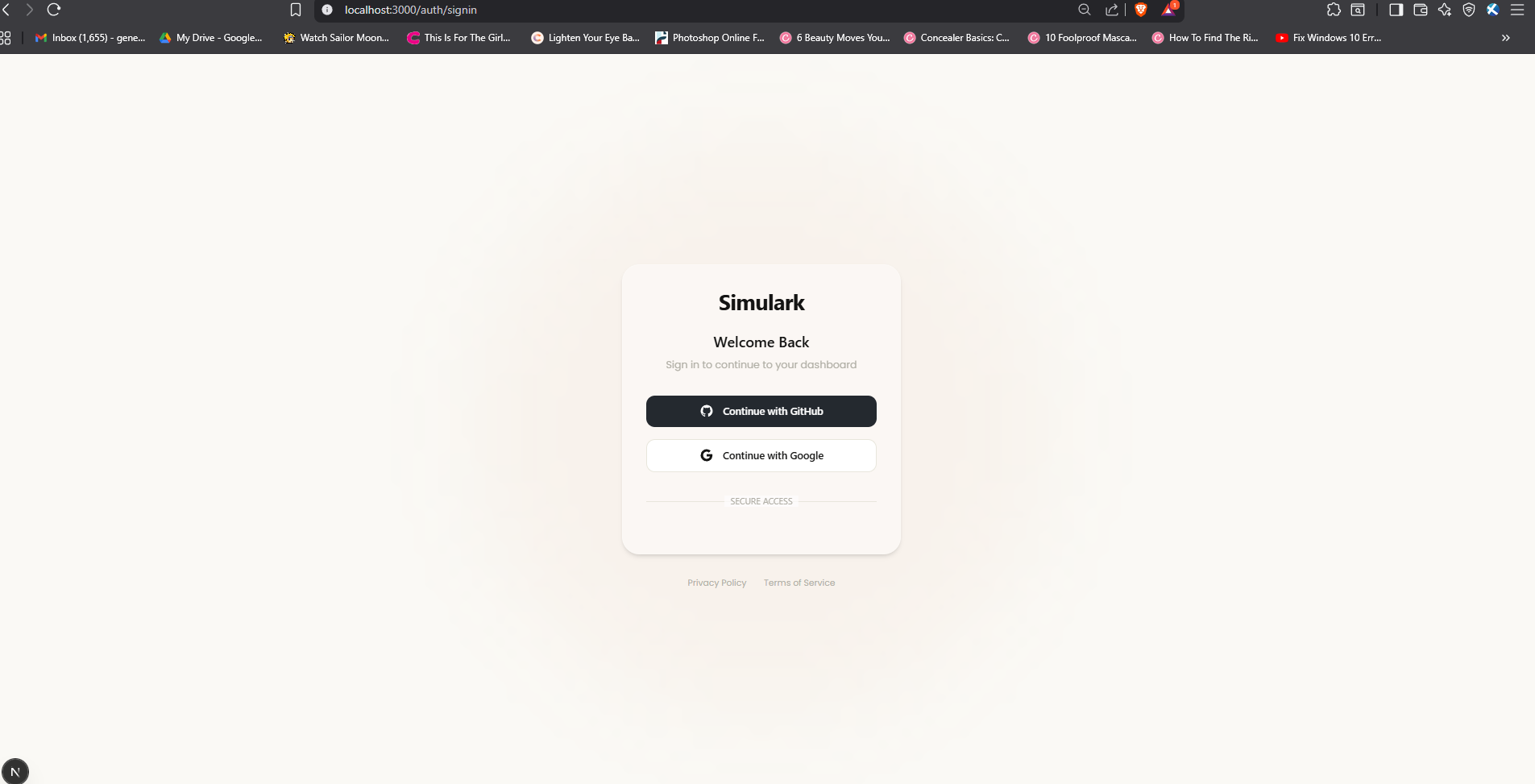




# Application Development and Testing

Implementation Details

## Frontend



#### User Authentication and Registration

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

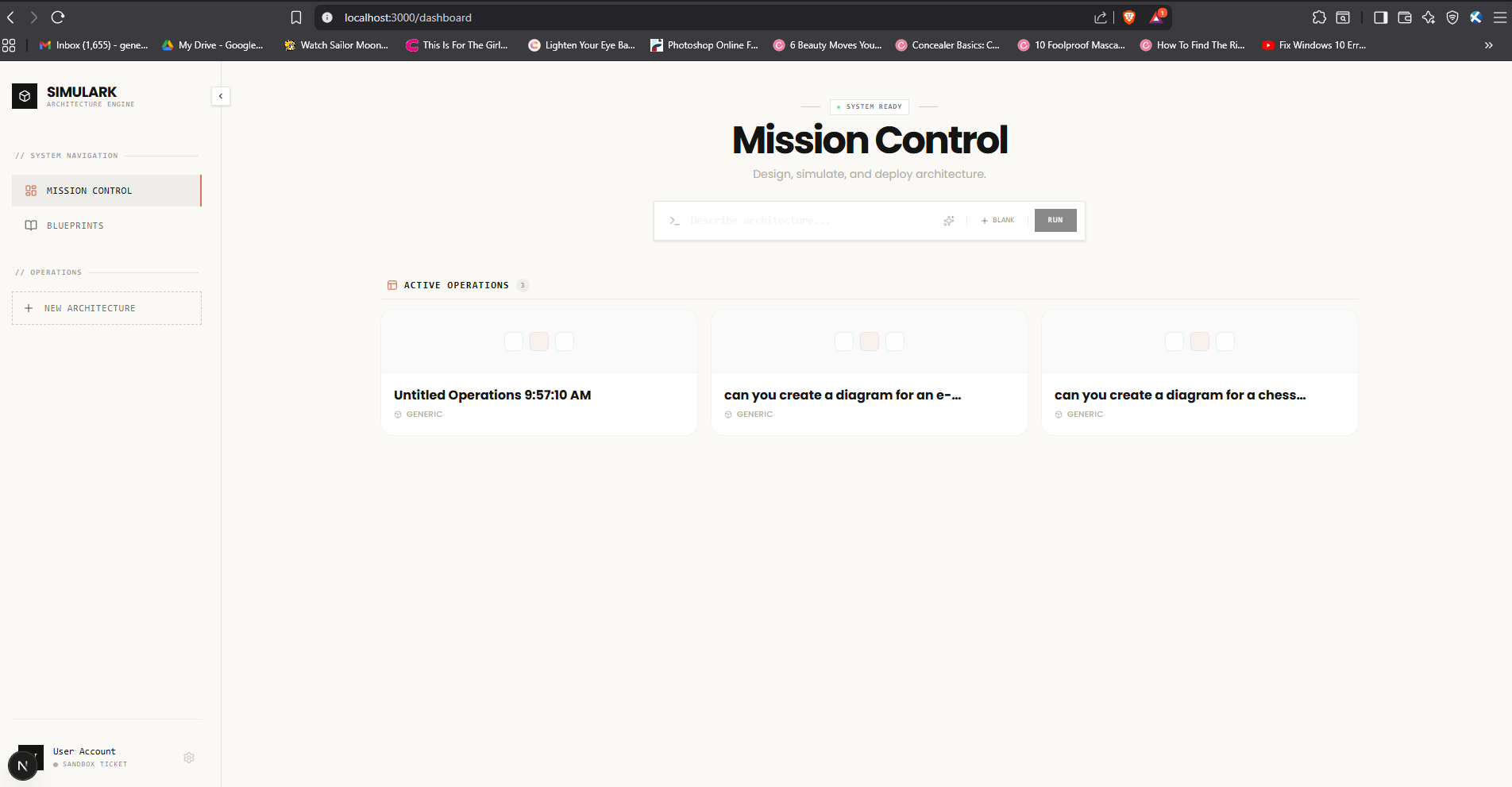
* **GitHub Login**: Dark button with GitHub icon for developers who prefer using their GitHub account
* **Google Login**: White button with Google icon for users with Google accounts
* **Error Banner**: Displays red error messages when authentication fails
* **Loading States**: Buttons show spinning 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:

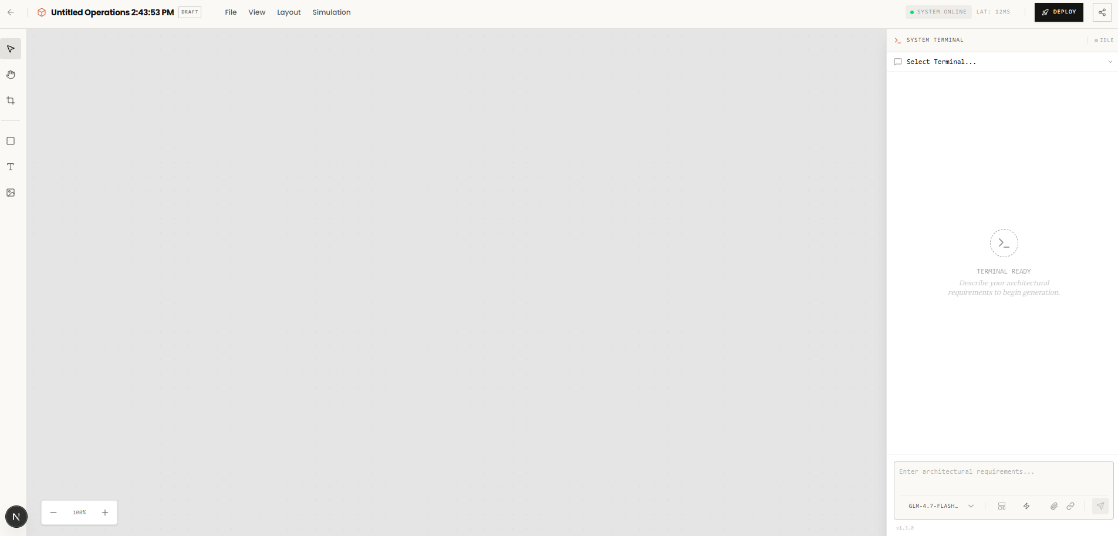
**Hero/Command Section**:

* + "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

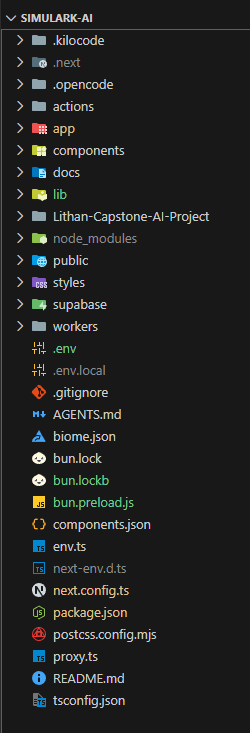
**7. Canvas Update** When the result arrives, the FlowEditor component receives the new nodes and edges and automatically:

* 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/)

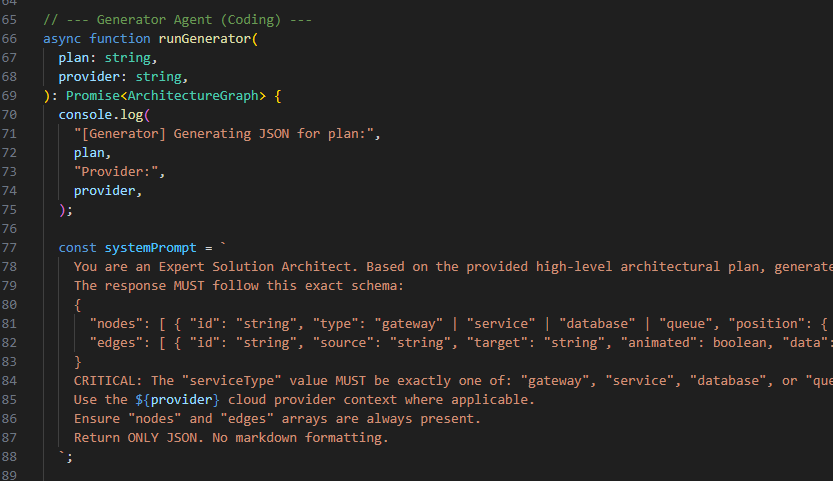
* Design guides, checklists, and reference documentation

**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)

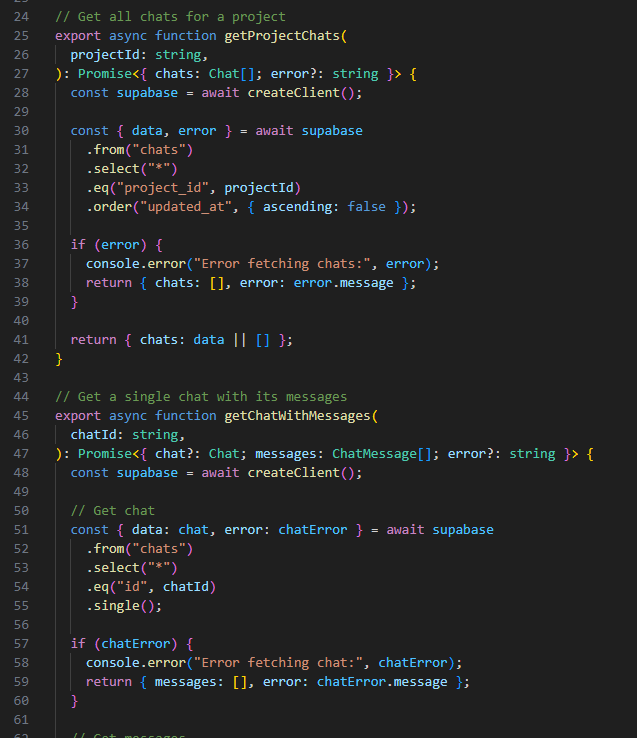
async function runGenerator(plan: string, provider: string): Promise<ArchitectureGraph>

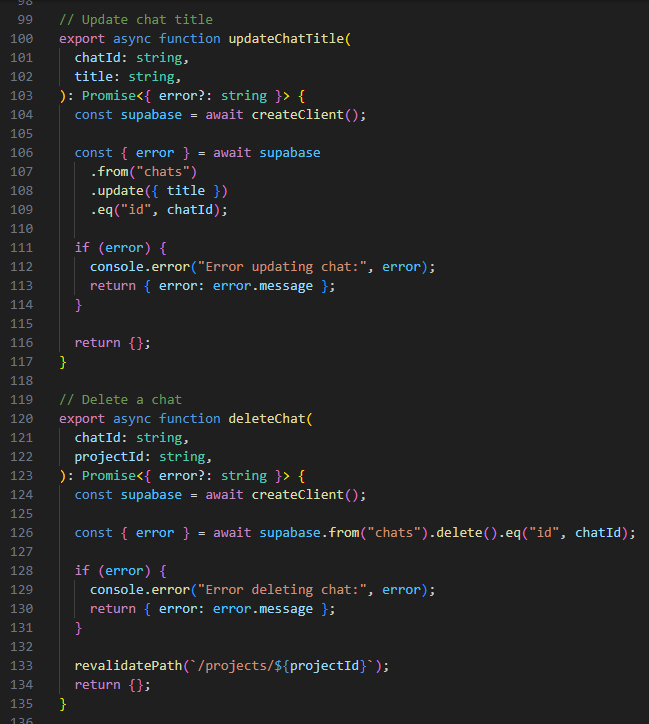
**Purpose:** Converts the aggregator's plan into structured JSON for the canvas.

**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)

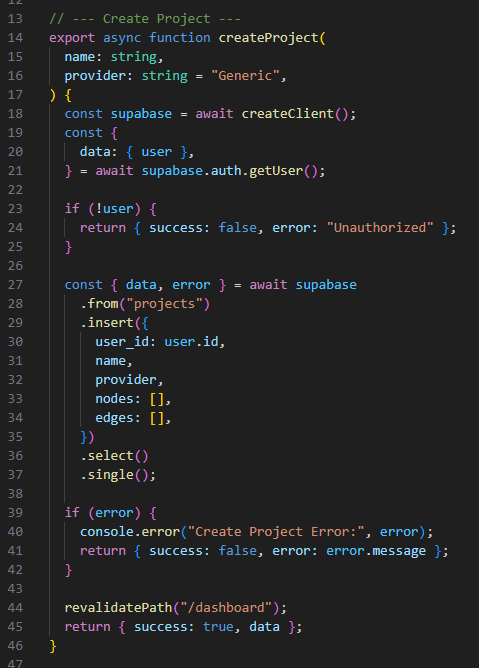
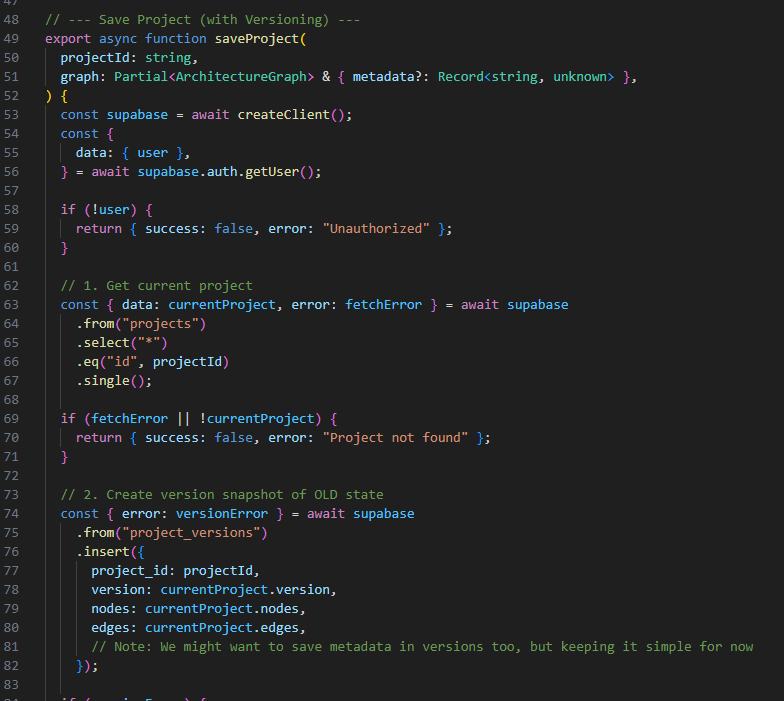
Chat System

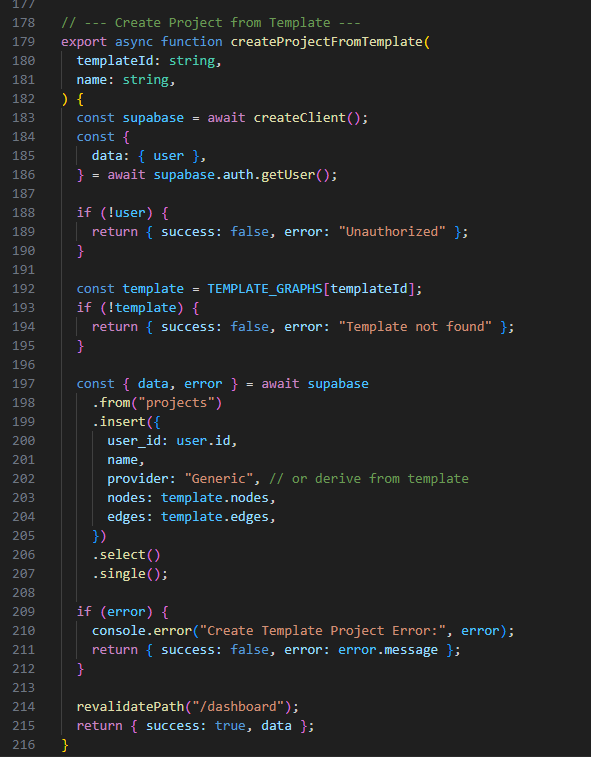
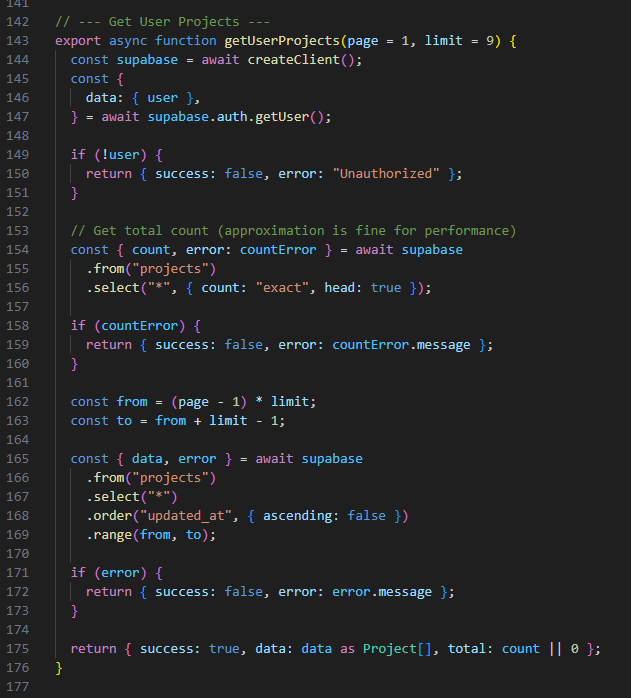
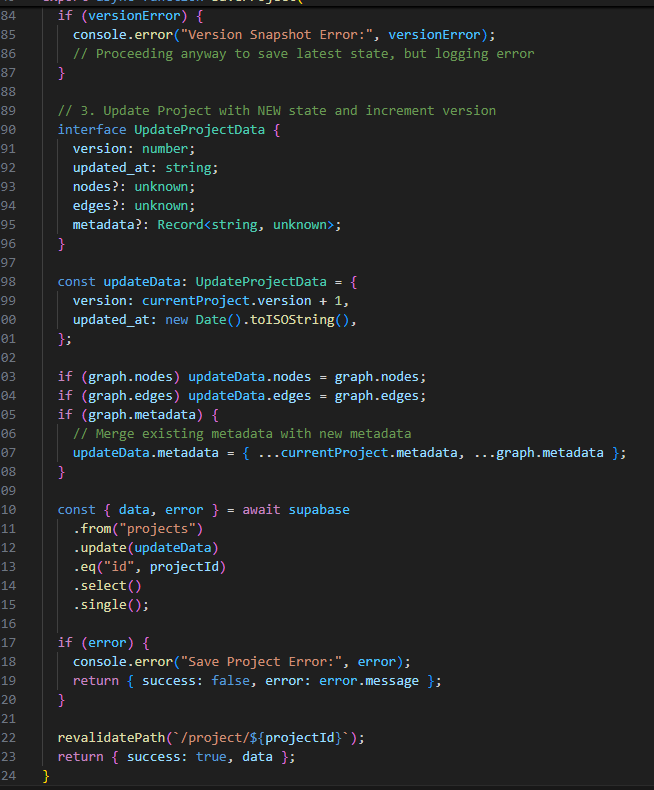
****



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

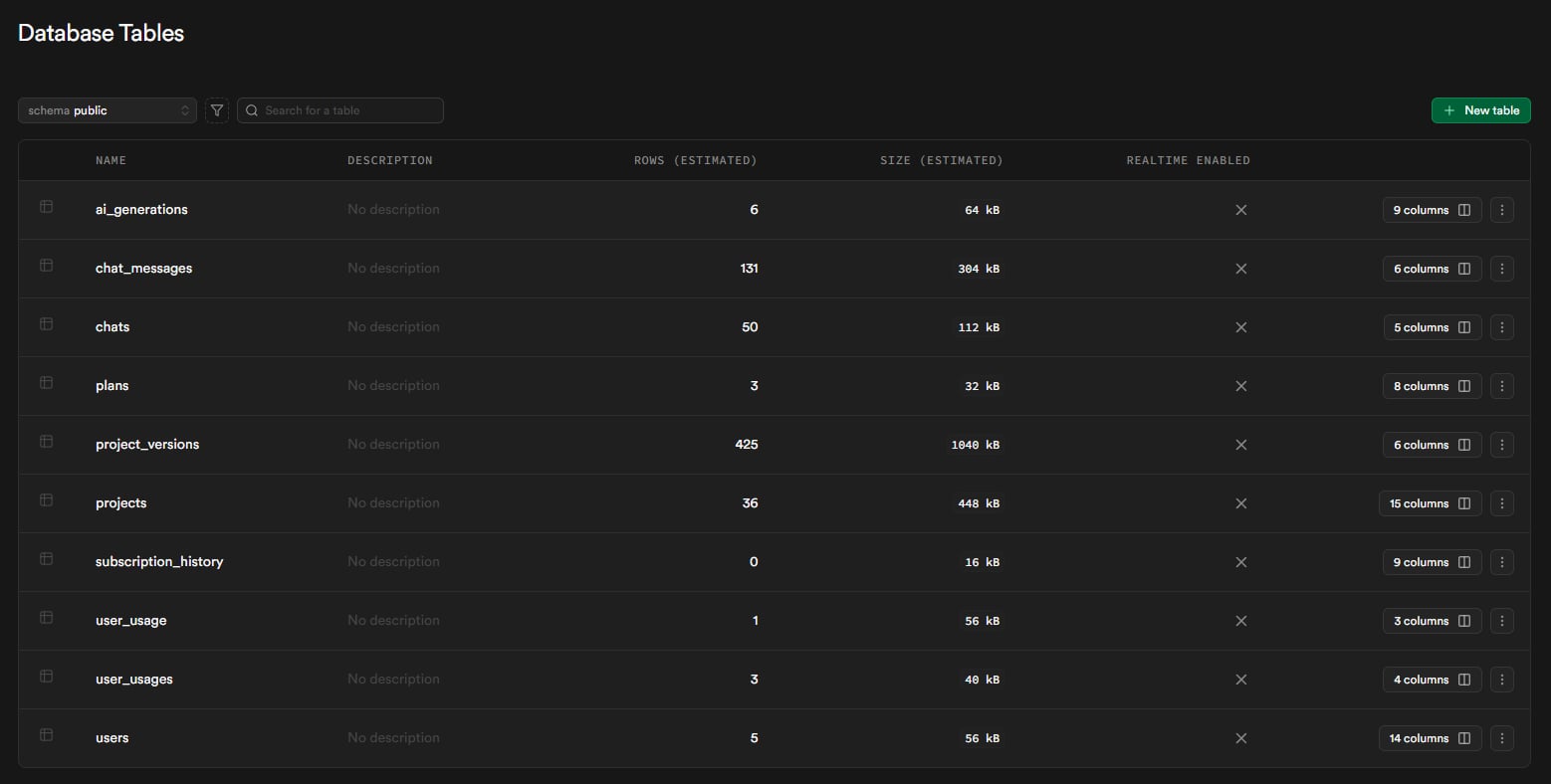
* projects - Main project storage (nodes, edges, metadata, version)
* project\_versions - Snapshots of previous states

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}

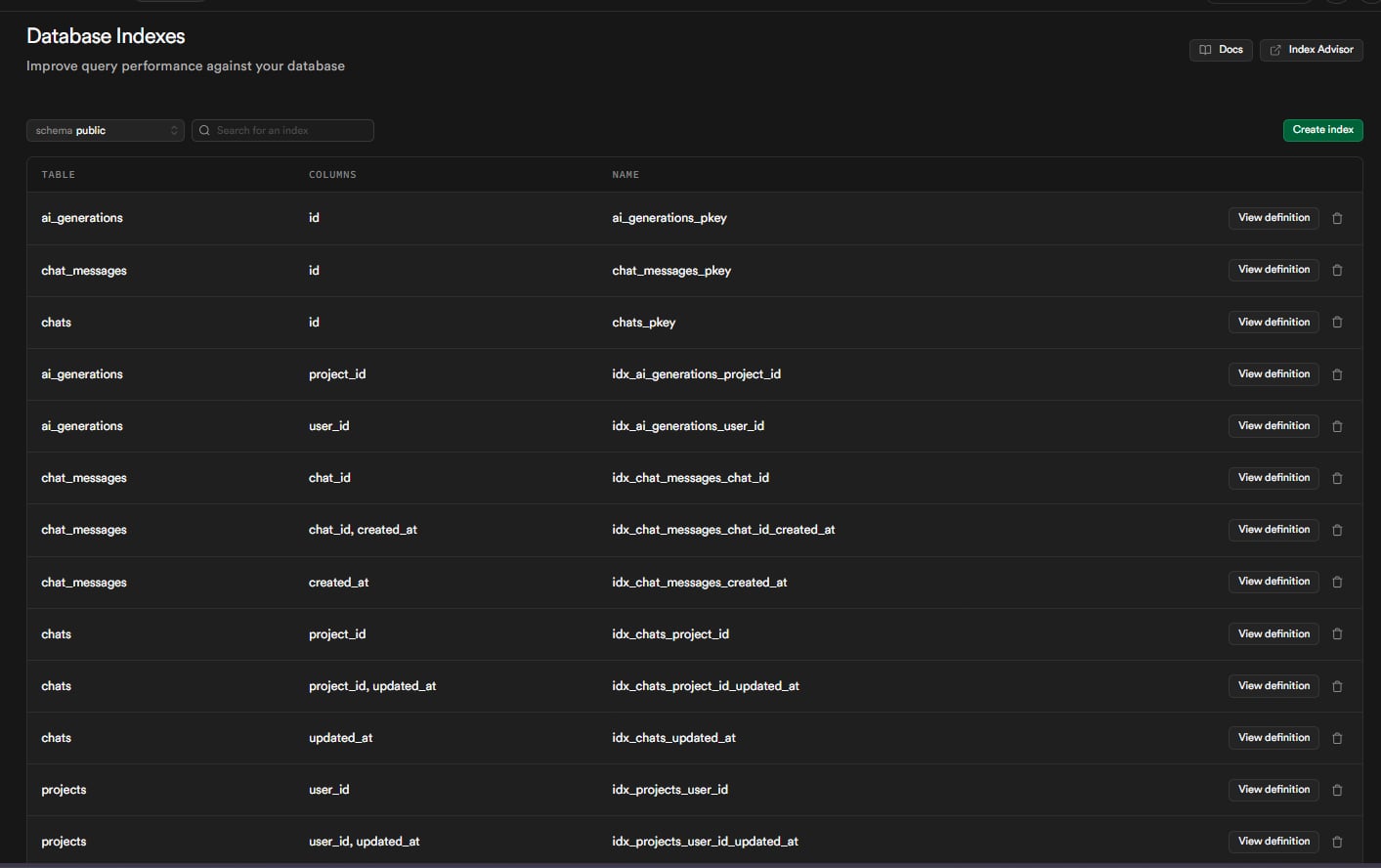
## Database Implementation

For backend services, the project uses **Supabase** (PostgreSQL) for persistent data storage, handling user authentication, project management, chat history, and AI generation tracking. Rate limiting and subscription management are enforced both at the application level and via PostgreSQL RPC functions. The platform integrates with **Upstash Redis** for caching and session management.



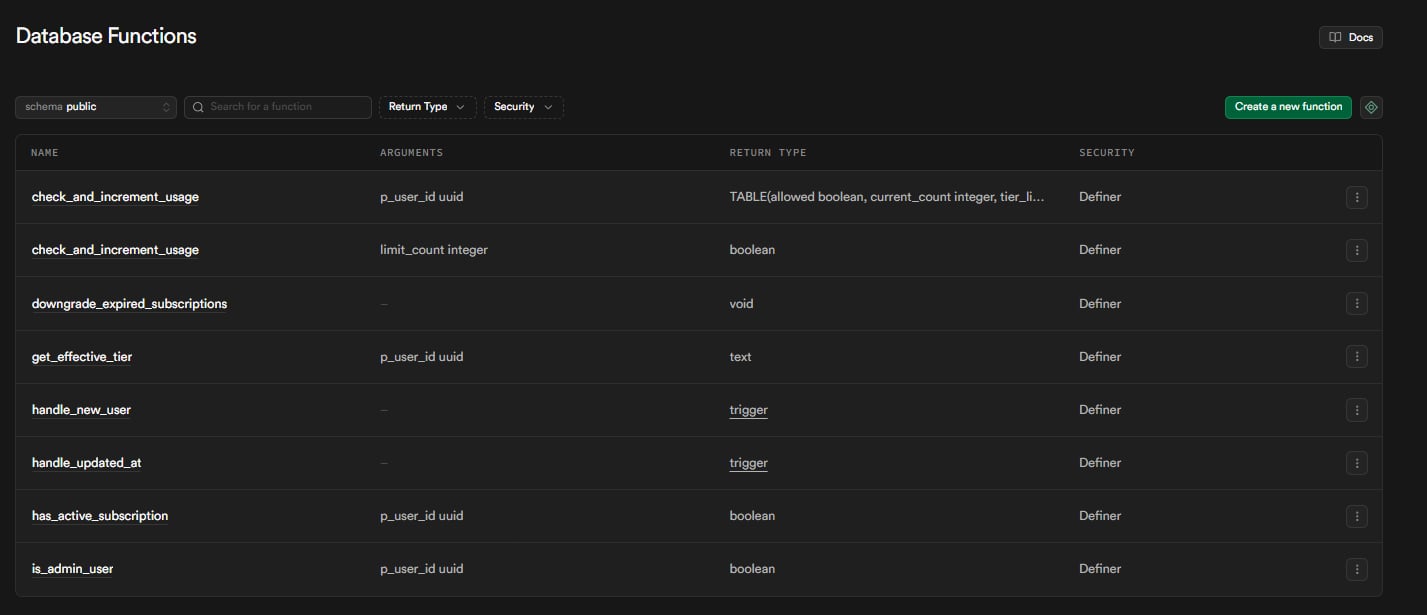
Database Tables

* **users**
  + Extends Supabase Auth for user profiles
  + Fields: user\_id, email, full\_name, avatar\_url, subscription\_tier, preferences
  + Auto-synced via trigger when new auth user is created
* **projects**
  + Stores architecture diagram projects
  + Fields: id, user\_id, name, description, provider, nodes, edges, is\_public, version, share\_token
  + 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**
  + Tracks AI model usage and success rates
  + Fields: id, user\_id, project\_id, prompt, model\_aggregator, model\_generator, success, tokens\_used
  + Used for analytics and debugging
* **user\_usage**
  + Daily rate limiting per user
  + Fields: user\_id, usage\_count, last\_used\_at
  + Checked via RPC function check\_and\_increment\_usage
* **chats**
  + Chat sessions per project
  + Fields: id, project\_id, title, created\_at, updated\_at
  + Multiple chats can exist per project
* **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\_usages** (alternative naming)
  + Daily generation limits tracking
  + Fields: user\_id, generation\_count, date, last\_updated
  + Resets automatically based on date comparison



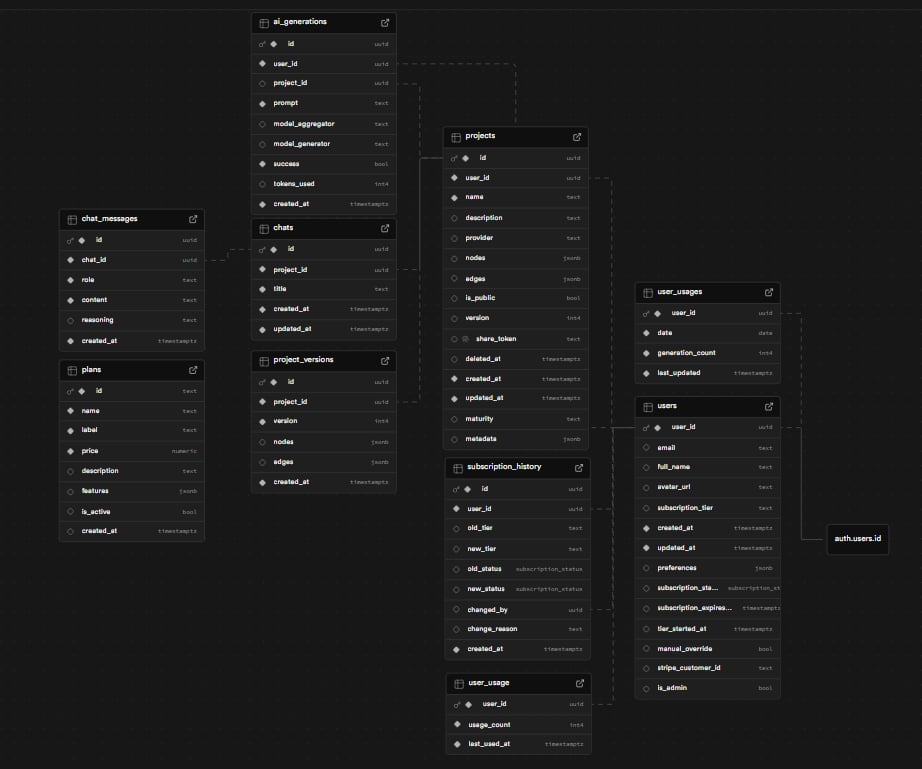
Database Indexes

* **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)
  + Rate limiting with tier-based limits (free: 10, pro: 100, enterprise: 1000)
  + Auto-creates usage record if not exists
  + 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)

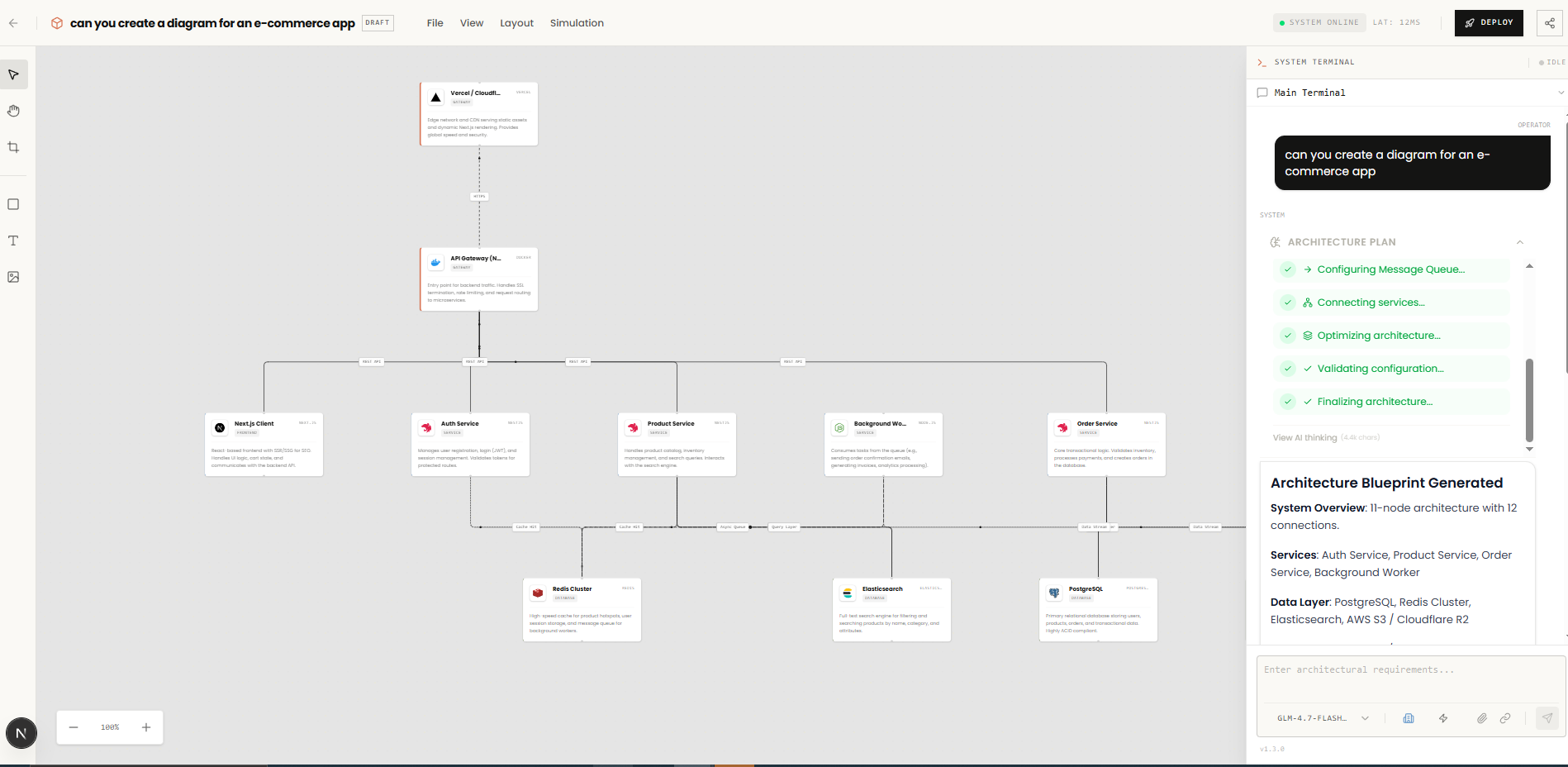




### Testing Methodology

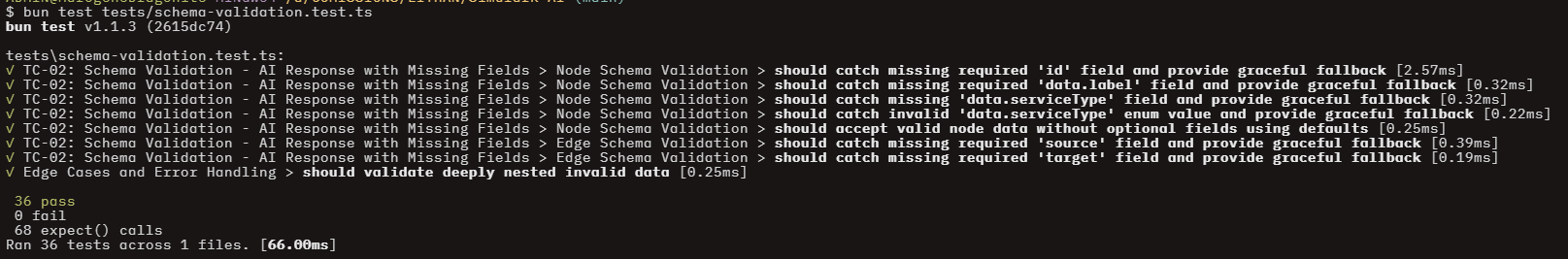
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Feature** | **Test Steps** | **Expected Results** | **Status** |
| TC-01 | AI Generation | Enter prompt “Create an e-commerce app diagram". Click Generate. | Returns valid graph with Gateway, Service, Database, Queue nodes | PASS |
| 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:



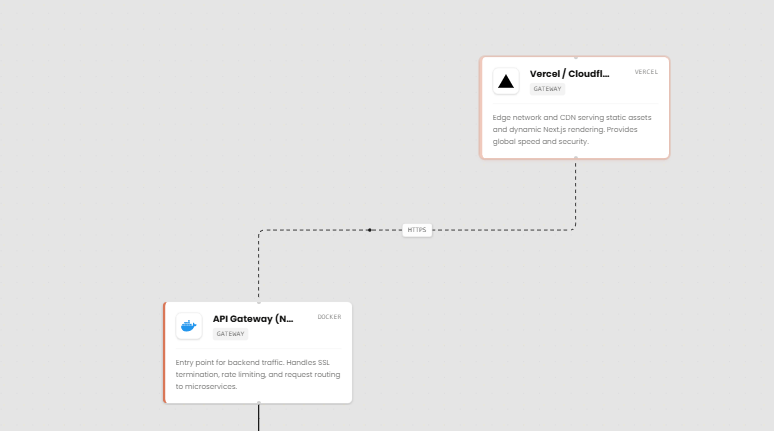
TC-01

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

 TC-02

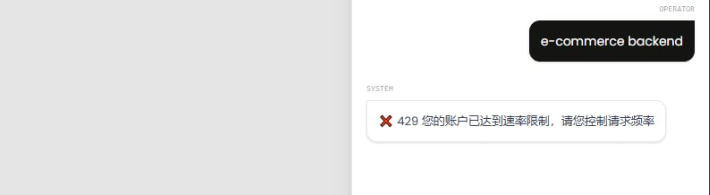
Tested missing fields using vitest and valibot



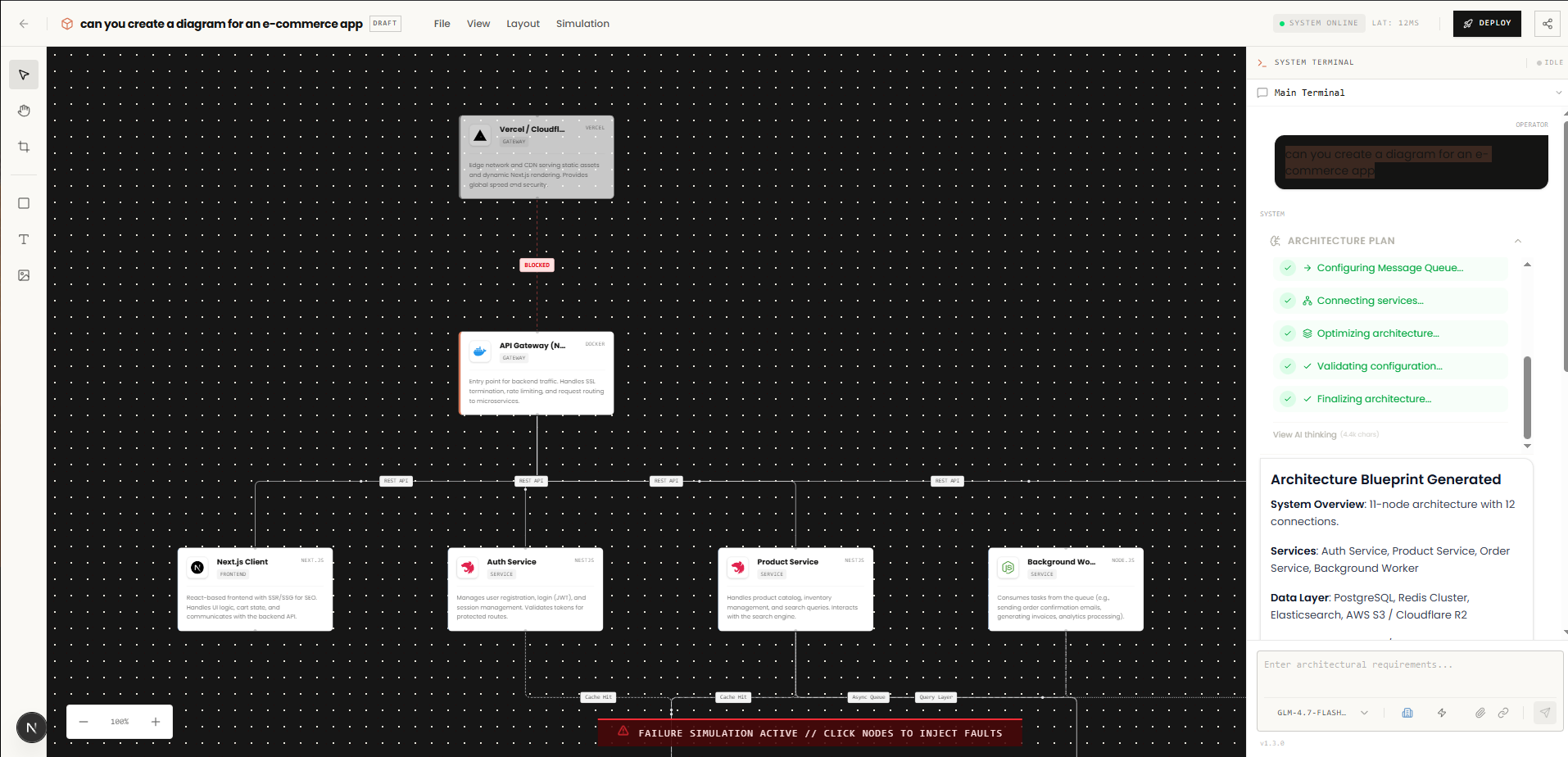


Can drag the figure and alter the connections between them

TC-04

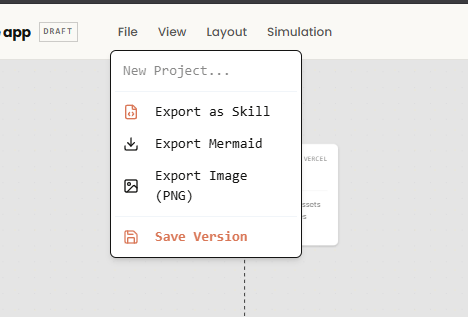


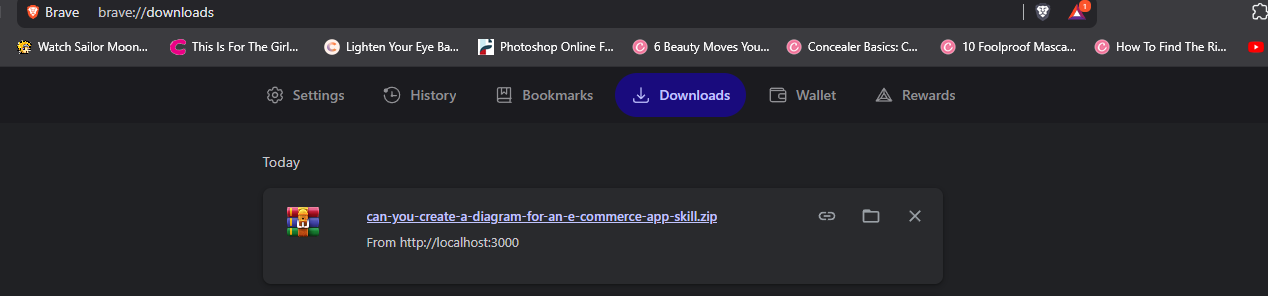
The user exceeded the limit of the free tier



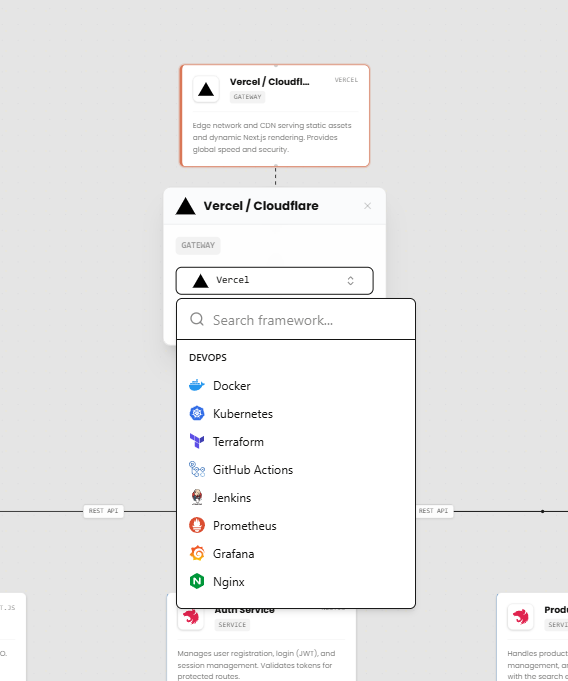
Traffic reroutes, visual feedback

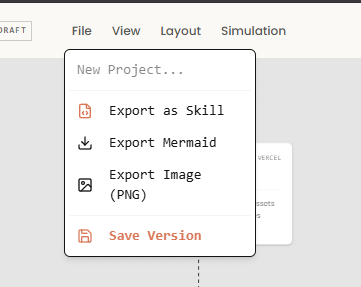
TC-06





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 correctly initializes multiple provider configurations (Zhipu, Kimi, Google, etc.), maps specific model IDs to their respective vendors, and maintains a reliable fallback chain.
* **Methodology:** A suite of automated unit tests was executed using a mock-driven strategy to validate configuration logic and error handling.
  + **Scope:** 18 distinct test cases across 6 functional categories.
  + **Provider Coverage:** Zhipu, OpenRouter, Kimi, Google, Minimax, and Anthropic.
  + **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:** The test suite achieved a **100% pass rate** across all 18 test cases, with an average execution time per test of approximately **0.11ms**. This indicates that the AI Client’s logic for provider routing and configuration is highly optimized and computationally lightweight. The successful validation of the **30-second timeout** and the **Zhipu-to-OpenRouter fallback chain** confirms that the system architecture is resilient and capable of maintaining service availability even if the primary provider fails. While the logs flagged a missing API key for Kimi, the "should handle missing API key gracefully" test passed, proving the model's error-handling logic is robust enough for production deployment.

**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:** The resilience suite achieved a **100% pass rate**, demonstrating that the system can distinguish between transient network issues and permanent "Bad Request" errors with high precision. The tests confirm that the **Exponential Backoff** logic correctly respects delay caps, preventing resource exhaustion during outages. Notably, the **AI Parsing** module successfully extracted structured data from unstructured markdown and raw text, which is critical for handling the inherent "noise" in LLM responses. With the **Circuit Breaker** integration validated, the system is proven to have a robust fail-fast mechanism that protects downstream services from cascading failures when a specific AI provider is underperforming.

**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: The Circuit Breaker achieved an outstanding 100% accuracy in state management, with transition latencies remaining below 0.1ms for most operations. This indicates that the system successfully identifies when a provider has crossed the failure threshold and immediately "trips" the circuit to block further doomed requests, thereby saving system resources and preventing cascading latency. The successful transition from Half-Open back to Closed following a successful probe call confirms that the recovery logic is sensitive and reliable. This makes the Circuit Breaker a vital safeguard for maintaining overall system availability when integrating with volatile third-party AI providers.

#### 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 Zod or JSON Schema validator (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: The validation suite achieved an impeccable 100% pass rate, confirming that the system’s "data contract" is strictly enforced across all infrastructure categories. By successfully rejecting invalid inputs—such as negative memory, zero replicas, and out-of-range AI temperatures—the schema layer acts as a critical first line of defense against malformed architectural configurations. The Mapping tests further ensure that the system is extensible, correctly falling back to a BaseSchema for unknown types rather than throwing unhandled exceptions. This robust validation layer ensures that downstream components can safely assume data integrity, significantly reducing the likelihood of runtime errors in the deployment pipeline.

#### 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: The prompt engineering suite achieved a 100% pass rate, demonstrating high precision in transforming raw user input into structured system instructions. The model successfully identifies a wide array of architecture types—from legacy monoliths to modern AI/ML pipelines—with specific logic to handle mixed architectures and ambiguous inputs. Notably, the validation layer is robust enough to filter out "noise" such as gibberish or empty greetings, while the System Prompt Building tests confirm that final outputs are properly constrained by complexity and framework-specific guidelines. This ensures that the AI receives a highly contextualized and high-quality prompt, significantly improving the accuracy of generated technical architectures.

#### 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).

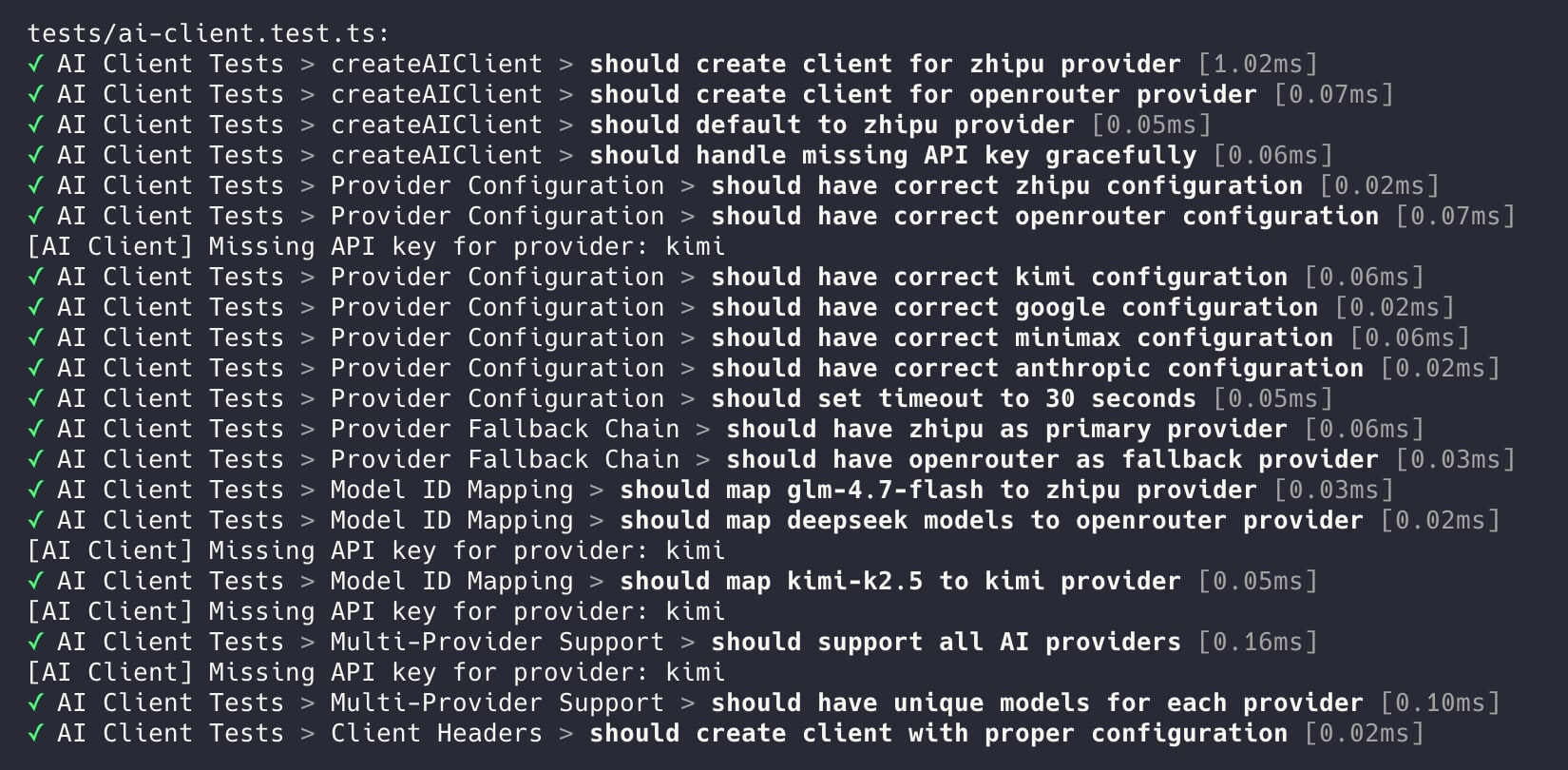
#### Resilience Strategy: Specifically tested against "poisoned" inputs, such as deeply nested invalid data and incorrect structural types (e.g., providing an array when an object is required).

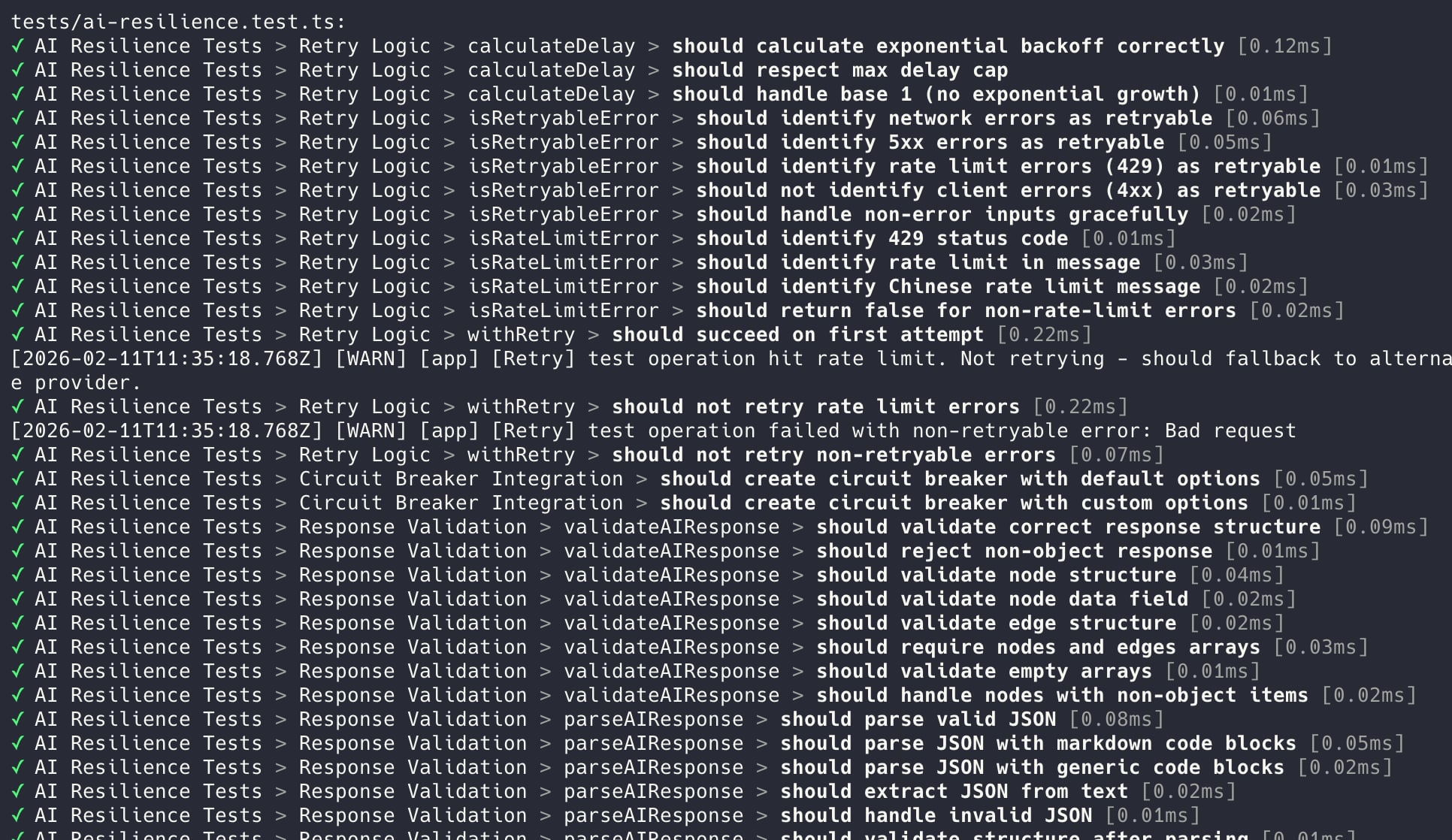
#### Environment: Comprehensive suite consisting of 185 tests passing in 64.00ms.

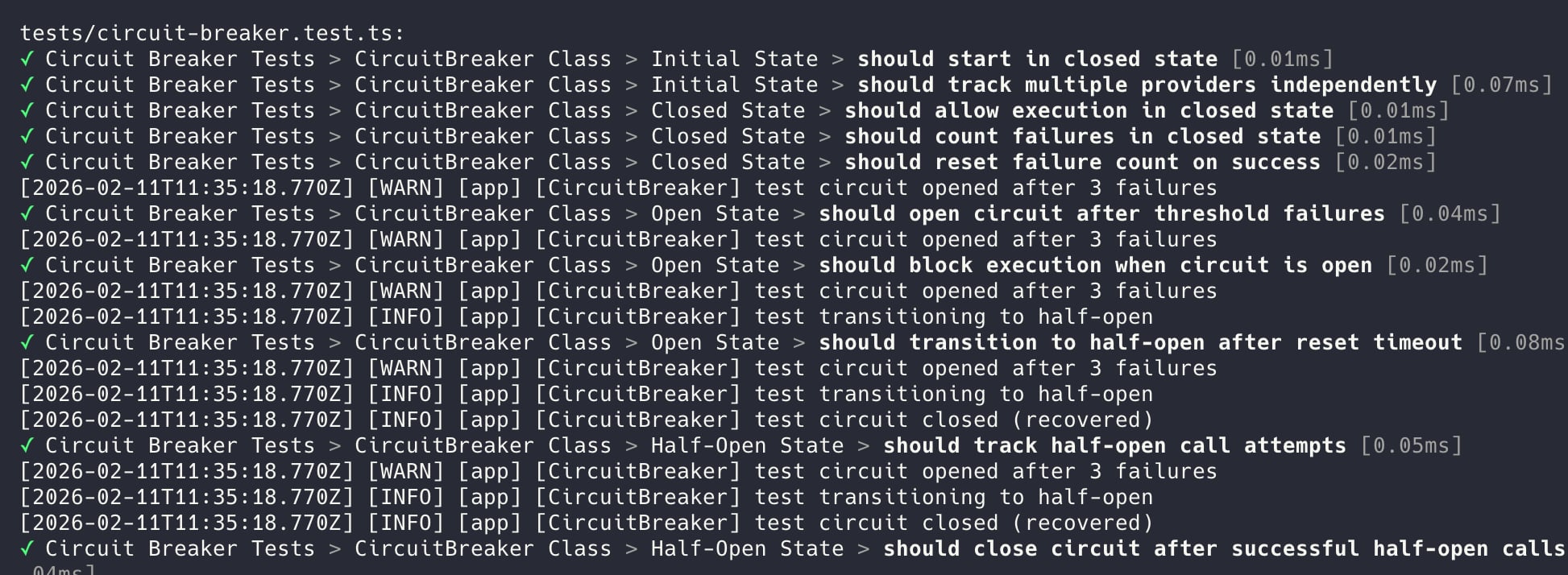
#### Metric: Data Sanitization Efficiency and Schema Enforcement Accuracy.

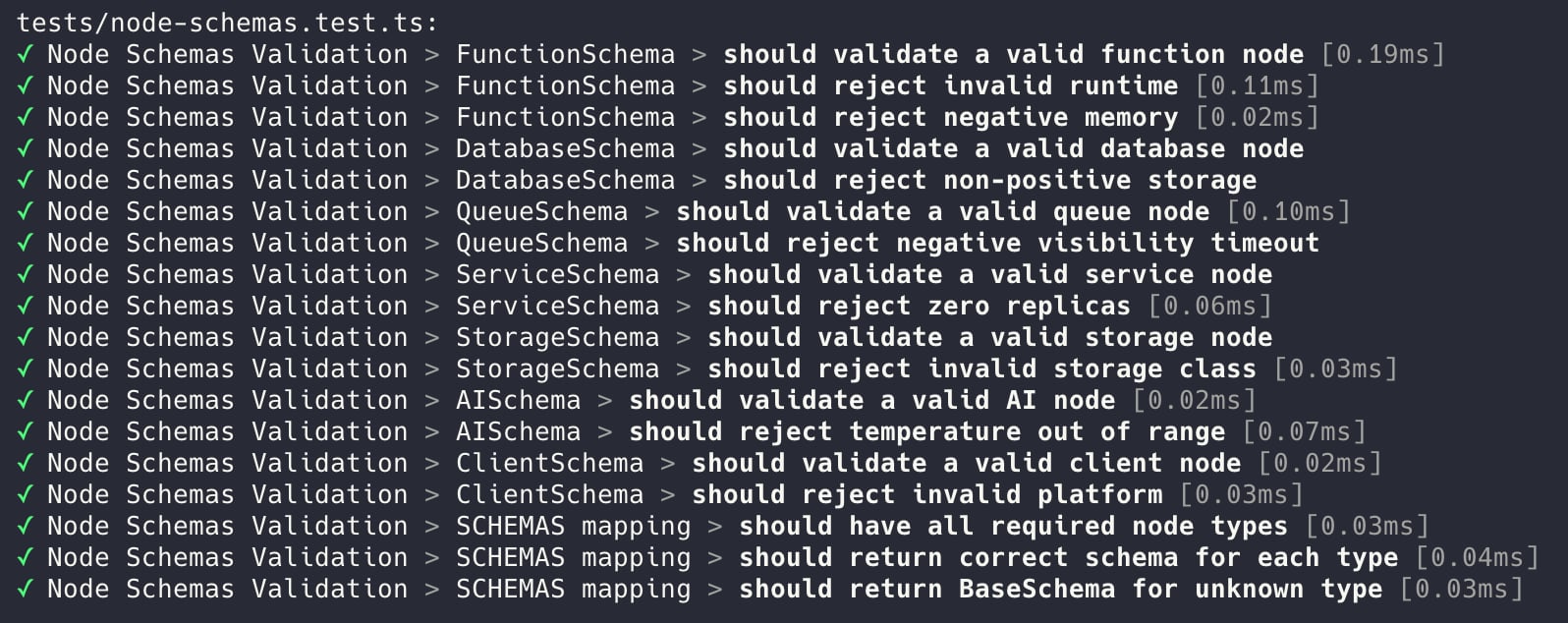
|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | Test Case | Status | Duration |
| 1. CreateProject Schema | Should validate minimal project creation | PASSED | 0.03ms |
|  | Should catch empty project name | PASSED | 0.03ms |
|  | Should catch project name exceeding max length | PASSED | 0.03ms |
|  | 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 |

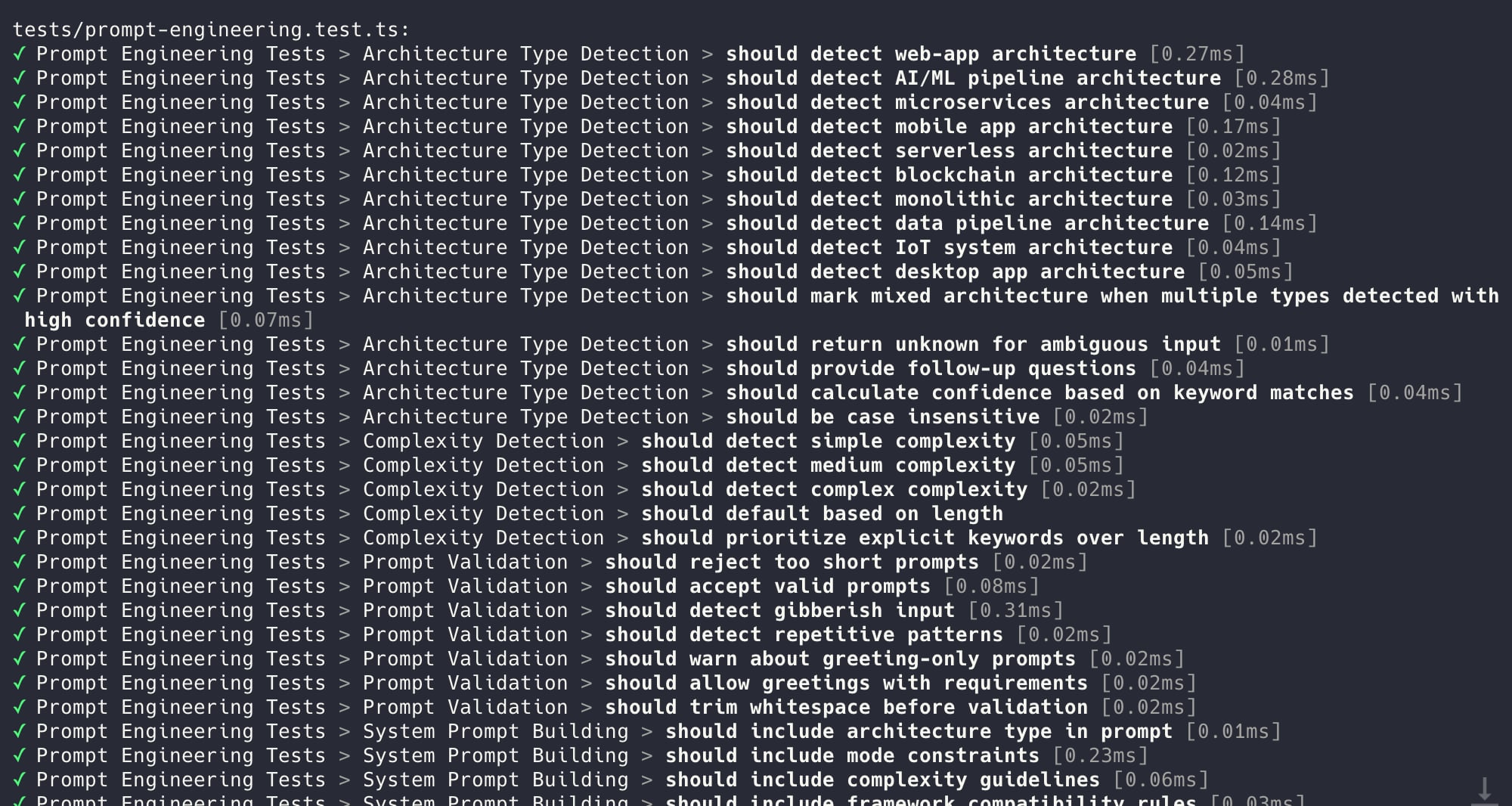
#### Analysis: The API Schema validation achieved a perfect pass rate, confirming that the gateway to the application is fully secured against malformed data. By enforcing strict constraints on UUID formats, email structures, and non-empty strings, the system prevents downstream database or logic errors caused by invalid primitives. The tests for the GenerateRequest schema are particularly vital, as they ensure that prompts—the core input for the AI—are neither empty nor incorrectly formatted before processing. Furthermore, the system's ability to catch deeply nested invalid data and handle null/undefined values gracefully indicates a mature, production-ready validation layer that can withstand both accidental user error and intentional edge-case exploitation.

****





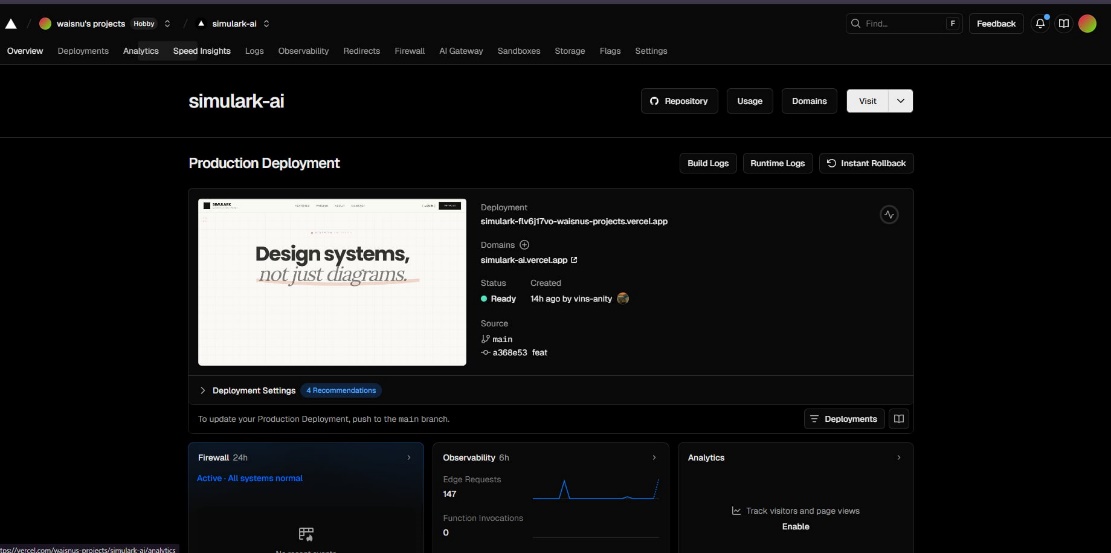


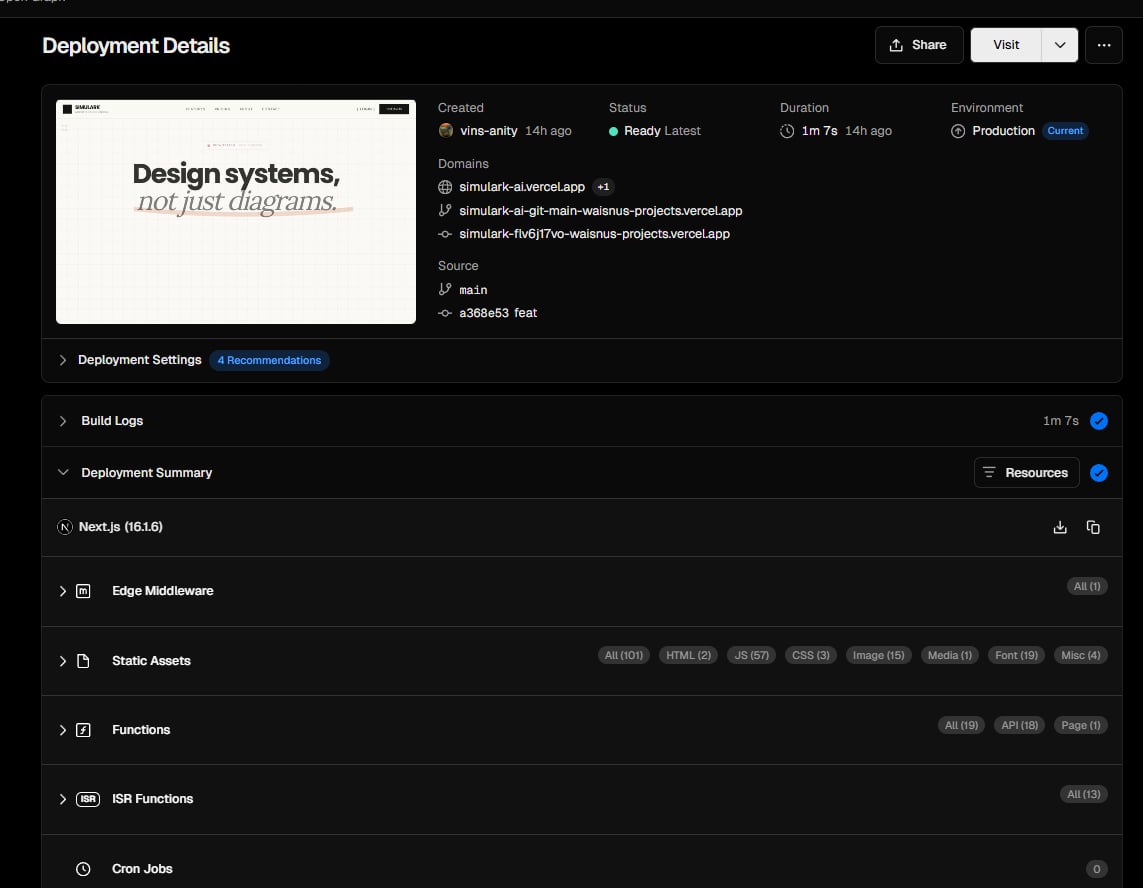


# 

# Deployment and Integration

#### Deployment Strategy

****

****

The Simulark AI project uses **Vercel** as its primary deployment platform with **Next.js 16** and **Bun** runtime. Key deployment characteristics:

**Infrastructure:**

* **Database**: Supabase (PostgreSQL) with Row-Level Security
* **Caching**: Upstash Redis for rate limiting
* **AI Services**: ZhipuAI (primary), OpenRouter (fallback)

**Security Strategy:**

* CSP headers with service allowlists
* Tiered rate limiting (AI: 10 req/10s, Auth: 5 req/min, API: 100 req/min)
* Comprehensive security headers (HSTS, X-Frame-Options, etc.)

**Deployment Flow:**

1. pnpm build → Next.js production build with Turbopack
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 Conclusion

**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 and Biome
* 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:**

* Enhanced AI model support (Claude, Gemini, Kimi)
* Collaborative editing with real-time sync
* Terraform/cloud deployment export
* Mobile responsiveness improvements
* Template marketplace and custom node SDK

# References

* **Core Framework & Runtime:** (Next.js 16, React 19, Bun, TypeScript)
* **Frontend Libraries:** (React, Tailwind CSS v4, Shadcn/UI, Radix UI, React Flow/ZYFlow, 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, Render, Redis Cloud)
* **UI & Styling:** (Tailwind CSS v4, CSS variables, clsx, tailwind-merge)
* **Key dependencies:** (date-fns, uuid, openai, @supabase/ssr, @t3-oss/env-nextjs)