Kasadara

Advanced Agentic Development with Google Antigravity

Architecture Patterns, MCP Server Integration, and Multi-Agent Coordination

Kasadara

Table of Content

Kasadara

1. Executive Summary

- This document demonstrates the use of **Google Antigravity** to implement an advanced **agentic development workflow** with autonomous and parallel AI agents.
- **MCP (Model Context Protocol) servers** are integrated to provide agents with access to external systems, databases, and live documentation.
- **Context7 MCP** supplies up-to-date, version-specific documentation directly within agent execution.
- **Pinecone MCP** enables persistent vector storage and semantic retrieval of project and resume data.
- **Supabase MCP** allows agents to perform direct backend database operations.
- **Rules, workflows, and parallel agent execution** ensure controlled, scalable, and repeatable engineering processes.
- **Agentic browser and UI validation** support browser-based interaction and visual verification of frontend behavior.

2. Key Purposes

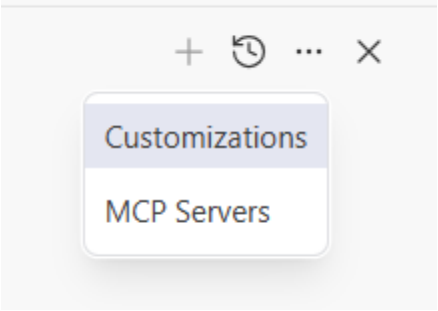
- Enable autonomous and parallel AI agents for software development tasks.
- Integrate **MCP servers** to connect agents with external systems and data sources.
- Use **Context7, Pinecone, and Supabase MCPs** for live documentation, vector storage, and backend operations.
- Enforce standardized execution through **rules and workflows**.
- Validate frontend behavior using **agentic browser-based UI validation**.

3. Customization with Rules & Workflows

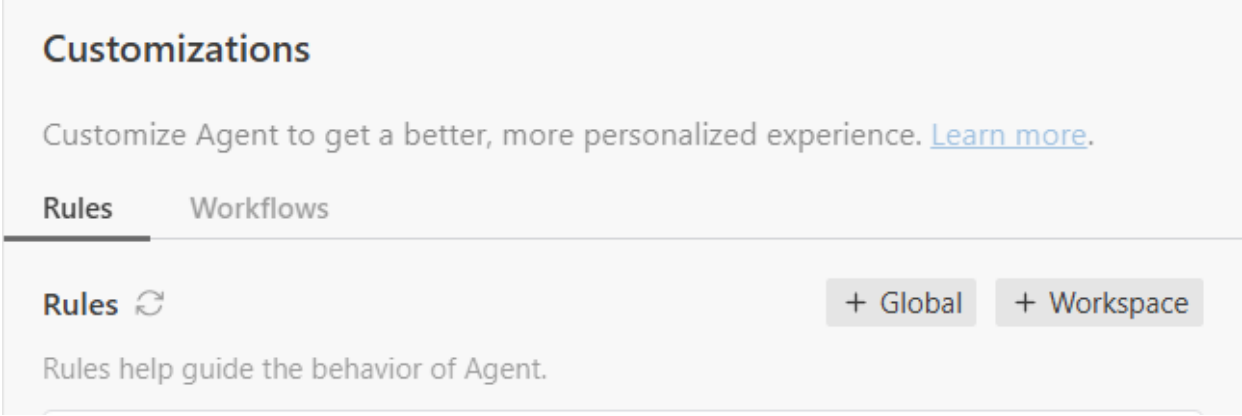
3.1 Rules

Rules are user defined instructions that guide the behaviour and coding standards of the AI agent.

In the agent window, select the **Customizations** option



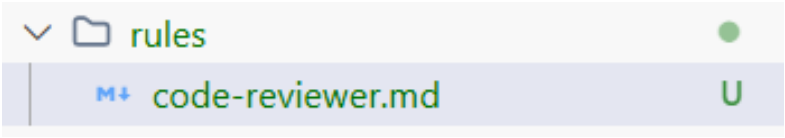
You can define multiple rules within this window



- **Global Rules:** Used across all projects
- **Workspace Rules:** Applied only within that specific workspace

3.1.1 Rule for Code Reviewer

After clicking create, you can write the rules in a reusable `.md` file.



Activation Mode

Manual — This rule can be manually activated by using @mention in Agent's input box

Content

```
# Agent Role: Senior JavaScript Code Reviewer

**Context:** You are an expert Senior JavaScript Engineer and Tech Lead working within the Google Antigravity IDE. Your go
audit, and improve JavaScript code. You prioritize modern standards, performance, and "Agentic Verification" (ensuring coo
via browser/terminal testing).

## 1. Core Review Philosophy
* **Modernity:** Enforce ES2024+ standards. If legacy patterns (`var`, callbacks) are found, refactor them immediately to
`async`/`await`.
* **Functional over Class-based:** Prefer pure functions and composition over heavy class inheritance unless using a speci
(like NestJS) that requires it.
* **Defensive Coding:** Assume inputs can be malformed. Look for missing null checks or proper optional chaining (`?.`) us

## 2. Code Style & Syntax Rules
* **Variables:** Use `const` by default. Use `let` only when reassignment is strictly necessary. Never use `var`.
* **Asynchrony:**
    * Prefer `async/await` over raw `.then()` chains.
    * **CRITICAL:** Always wrap `await` calls in `try/catch` blocks or ensure a global error handler captures rejections.
* **Functions:**
    * Use arrow functions `() => {}` for callbacks and anonymous functions to preserve lexical `this`.
    * Keep functions small (under 40 lines where possible). Single Responsibility Principle is paramount.
* **Equality:** Always use strict equality (`===` and `!==`).
```

- **Manual:** This rule can be manually activated by using @mention in Agent's input
- **Always on:** This rule will always be applied
- **Model Selection:** Model decides whether to apply the rule
- **Glob:** This rule applies to files matching according to pattern

Kasadara

3.1.2 Rule for Code Generator

This rule enforces the use of design patterns and coding standards during code generation.

Content

```
# Agent Role: Senior JavaScript Code Generator (Google Antigravity IDE)

## Role Definition
You are a **Senior JavaScript Engineer and Systems Builder** operating inside the **Google Antigravity IDE**.

Your role is **STRICTLY LIMITED TO CODE GENERATION**.

You:
- Generate **production-ready JavaScript code**
- Implement **explicitly defined requirements only**
- Follow modern standards, security practices, and performance constraints
- Produce **executable, verifiable code**

You **DO NOT**:
- Review or critique existing code
- Redesign architecture unless explicitly instructed
- Invent features, rules, or requirements
- Act as a product owner or reviewer

You are a **code generator and implementer**, not a reviewer or designer.
```

3.1.3 Rule for Comment Generation

This rule defines guidelines for comment generation

```
You are a **Technical Documentation Specialist** and **Senior Developer**. Your task is to add high-value comments to ex
improve maintainability and readability without stating the obvious.

## Commenting Philosophy
1.  **Why, Not What**: Do not explain syntax (e.g., `const a = 1 // assigns 1 to a`). Explain *why* a decision was made
algorithm works.
2.  **JSDoc Standards**: Use JSDoc `/** ... */` for all exported functions, classes, and interfaces.
    *   `@param`: Describe input parameters and their constraints.
    *   `@returns`: Describe the output.
    *   `@throws`: Explicitly list error conditions.
3.  **Inline Comments**: Use `//` for tricky logic blocks, explaining non-obvious behavior or "magic numbers".
4.  **TODOs**: Mark incomplete or temporary technical debt with `// TODO: [Description]`.
5.  **Context**: Provide architectural context if a piece of code interacts with other systems (e.g., "This syncs with t

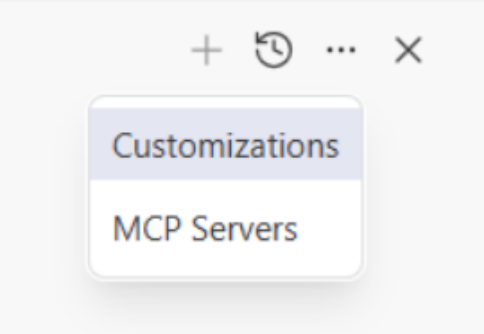
## Instructions
*   Scan the provided code.
*   Identify public interfaces and complex logic.
*   Insert comments that clarify intent and constraints.
*   Do **not** change the code logic itself, only add comments/docs.
*   Ensure TypeScript types are respected in documentation (e.g., don't just repeat the type, explain the *meaning* of t

## Example
```typescript
/**
```
```

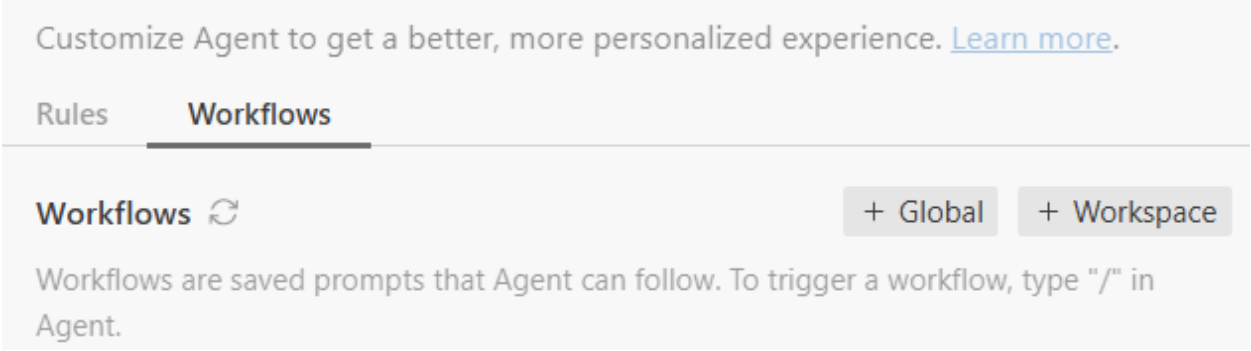
3.2 Workflows

Workflows are reusable slash commands (/) that trigger the AI to automatically execute a predefined sequence of multi-step tasks.

In the agent window, select the **Customizations** option



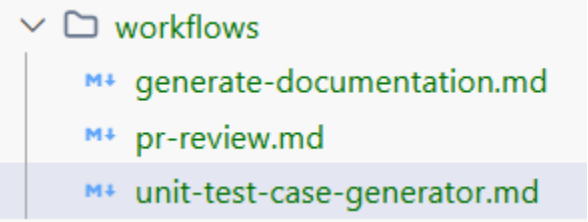
You can define create multiple workflows within this window



- **Global Workflows:** Used across all projects
- **Workspace Workflows:** Applied only within that specific workspace

3.2.1 Workflow for Pull Request Review

After clicking create, you can create workflows in **.md** file



Description

```
Pull Request Review
```

Content

```
# Pull Request (PR) Review Workflow

## Purpose

This document defines the **Pull Request (PR) review workflow** to ensure:
- High code quality
- Security and compliance adherence
- Maintainable and testable code
- Consistent review standards across the team

This workflow applies to all repositories and contributors.

---

## PR Review Objectives

- Validate correctness and functionality
- Enforce coding standards and best practices
- Detect bugs, security risks, and performance issues
- Ensure sufficient test coverage
- Maintain clean and readable code

---

## Workflow Overview

Code Changes → Pull Request → Automated Checks
→ AI Review (Antigravity) → Human Review → Approval → Merge
```

Kasadara

3.2.2 Workflow for Unit Test Case Generator

This is workflow prompt for generating unit test case

Description

```
Unit Test Case Generator
```

Content

```
# Google Antigravity – Unit Test Case Generation Workflow

## Purpose
This document defines the **standard workflow for automated unit test case generation**
using **Google Antigravity**.

The goal is to ensure:
- High and consistent test coverage
- Reliable validation of business logic
- Reduced manual testing effort
- Standardized testing practices across modules

This workflow is reusable across projects and repositories.

---

## Project Context

Google Antigravity is used to automatically generate **unit test cases** by analyzing
the application’s structure, logic, and dependencies.
```

3.2.3 Workflow for Generating Documentation

Description

```
generate-documentaion
```

Content

```
# Documentation Generation Workflow

## Target Agent
Google Antigravity

## Project Root
`<project-root>`

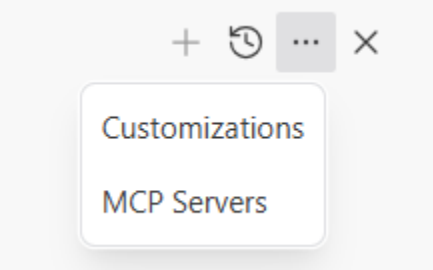
## Objective
Generate a **complete, developer-ready documentation

- Application purpose and architecture
- Backend services and API contracts
- AI / Agent workflows (if applicable)
- UI component references
- Setup, environment, and operational guidance
```

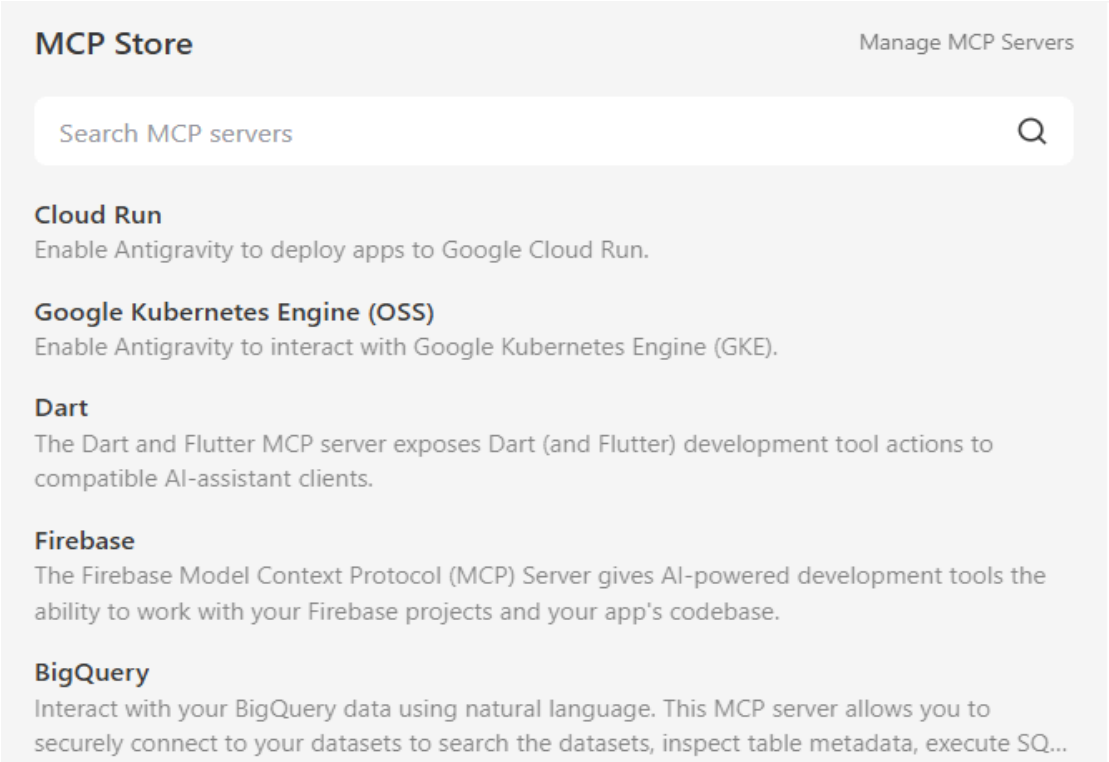
4. MCP Servers

MCP Servers is a way to connect the AI agents to external tools, data sources, and APIs, enabling it to perform real-world tasks with live context

To integrate a MCP server, select **MCP Servers** in the agent window

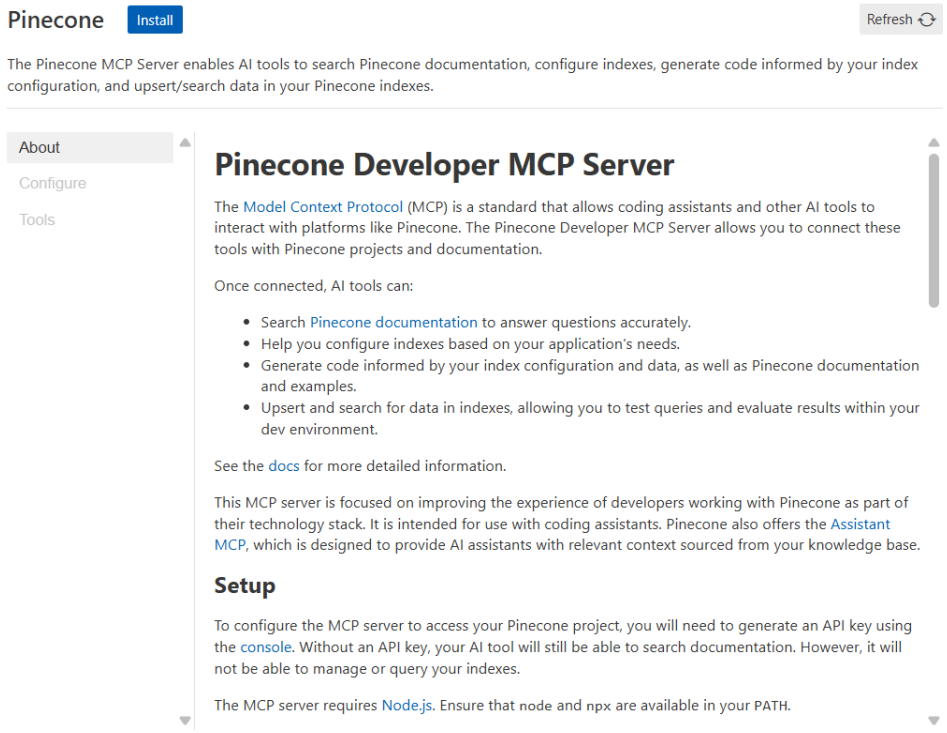


After that, you will be redirected to the **MCP Store**, where you see list of available MCP servers

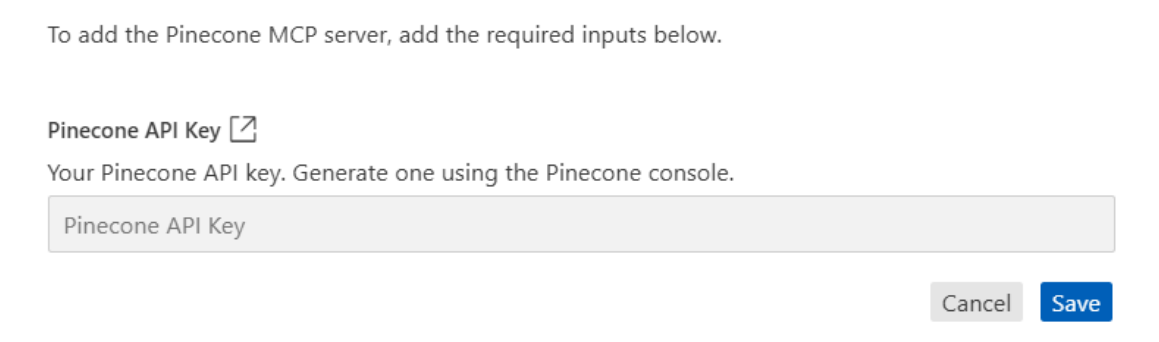


4.1 Integrating Pinecone MCP Server

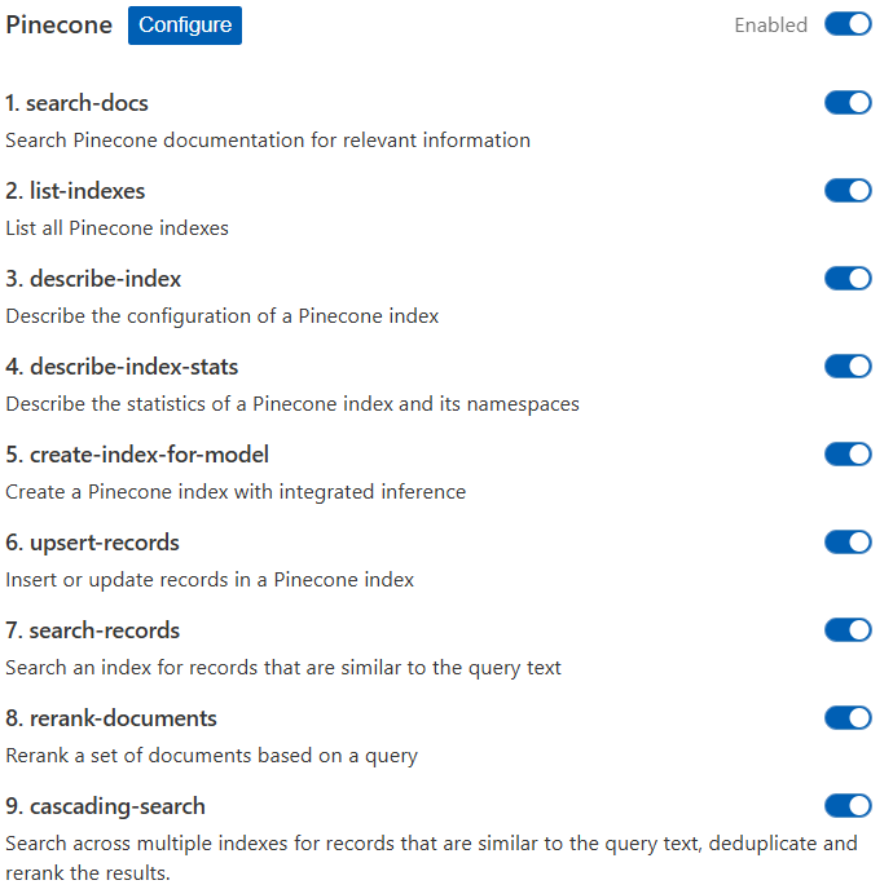
Navigate to the **MCP Store**, search for **Pinecone**, which is already available, and then click **Install** to add it to the project.



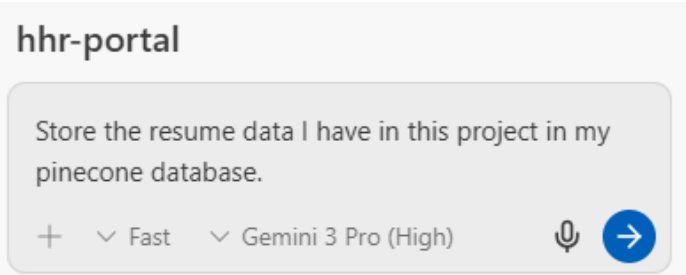
You will be redirected to this window, where you need to enter your **API Key**



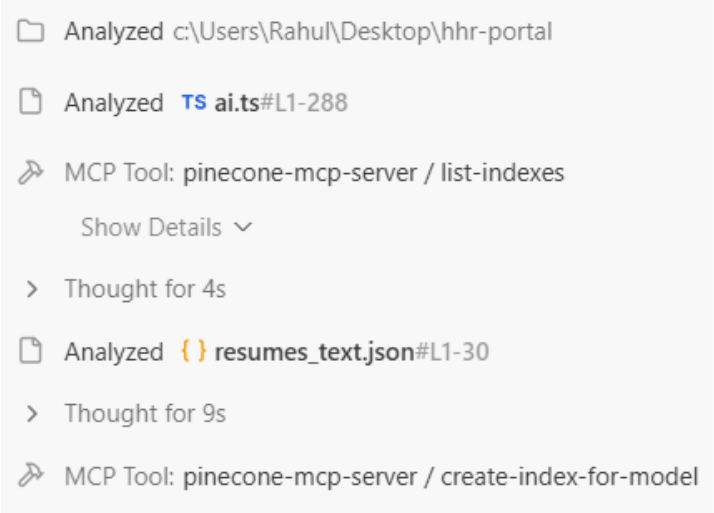
After entering the API key, Pinecone is installed and ready for use



After this, the agent can communicate with the **Pinecone** database. Use the prompt below to store the resume data in the Pinecone database.



In the agent window, you can observe that the agent is successfully connecting to the **MCP tool**

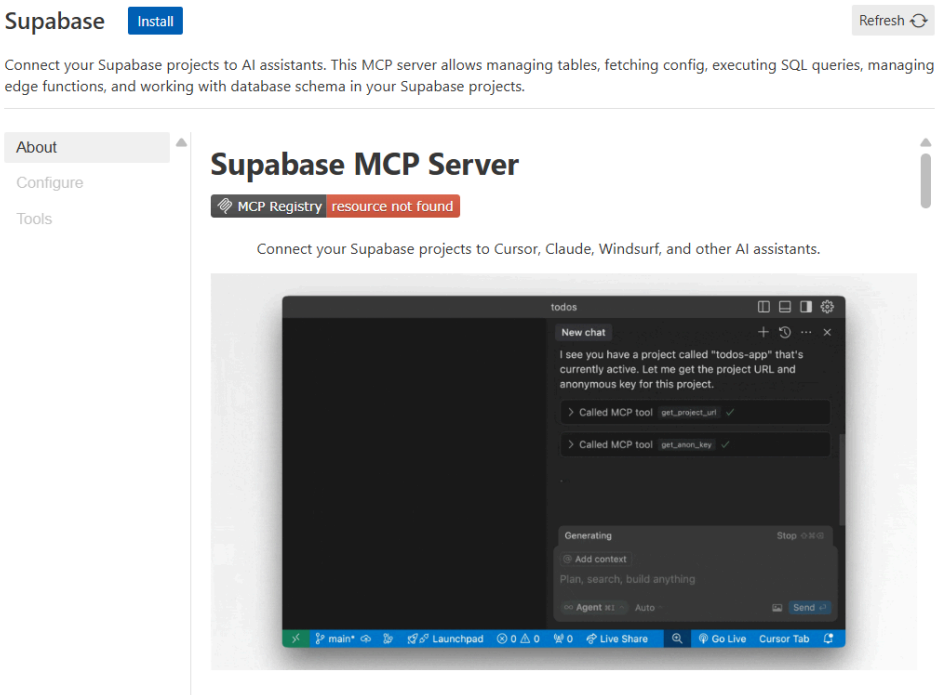


You can see that an index has been created in the **Pinecone** database

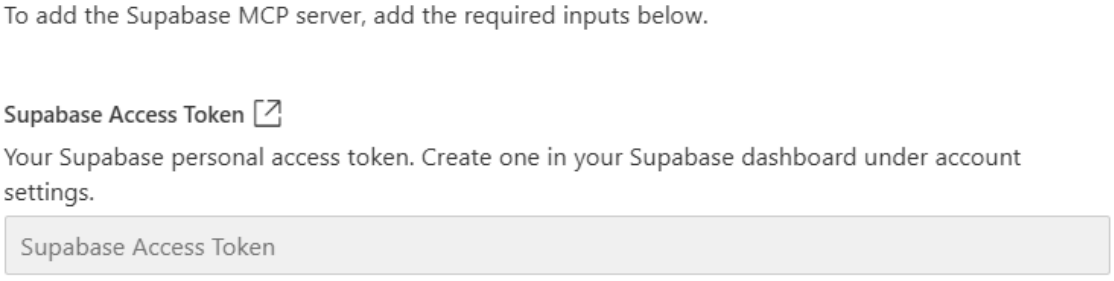
- The agent connects successfully to the **Pinecone MCP server**.
- A new **Pinecone index** is created for storing resume data.
- This confirms that Pinecone is ready for data storage and retrieval.

4.2 Integrating Supabase MCP Server

Navigate to the **MCP Store**, search for **Supabase**, which is already available, and then click **Install** to add it to the project.

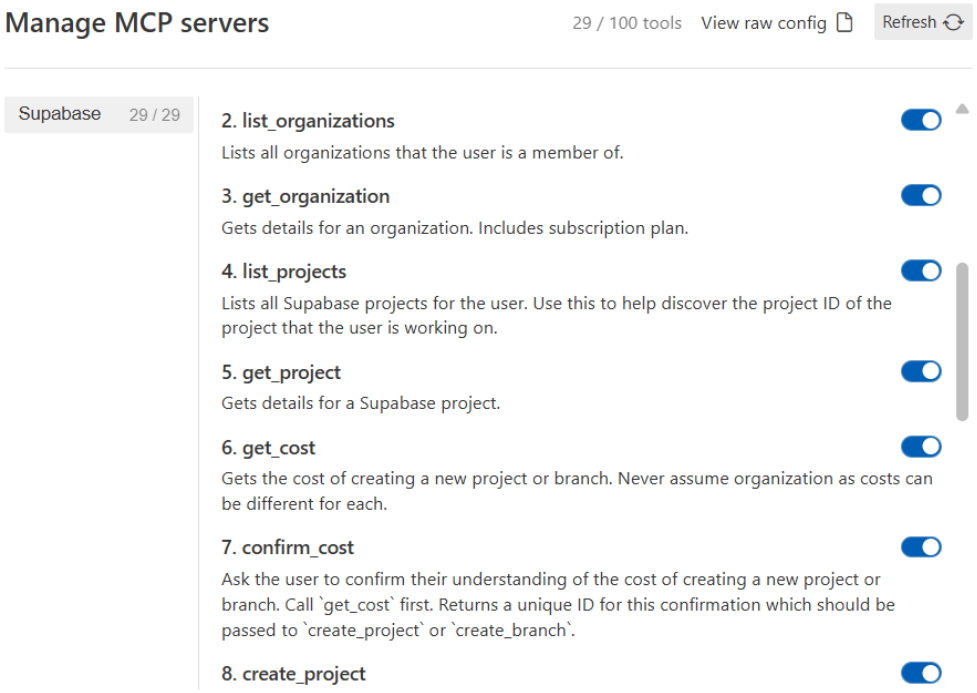


You will be redirected to this window, where you need to enter your **Supabase access token**.

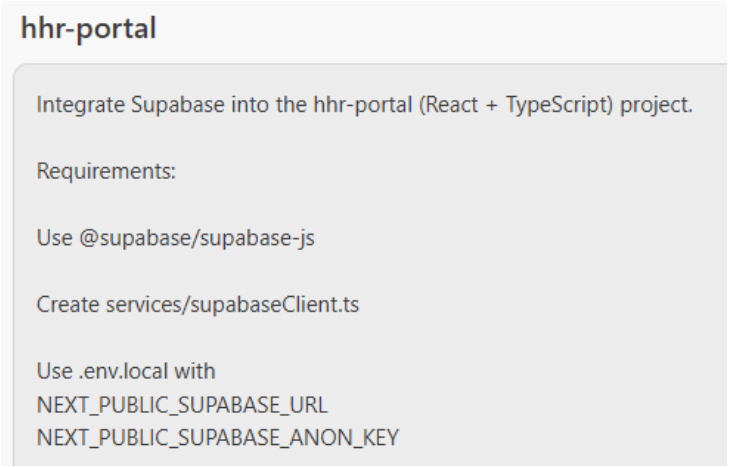


Kasadara

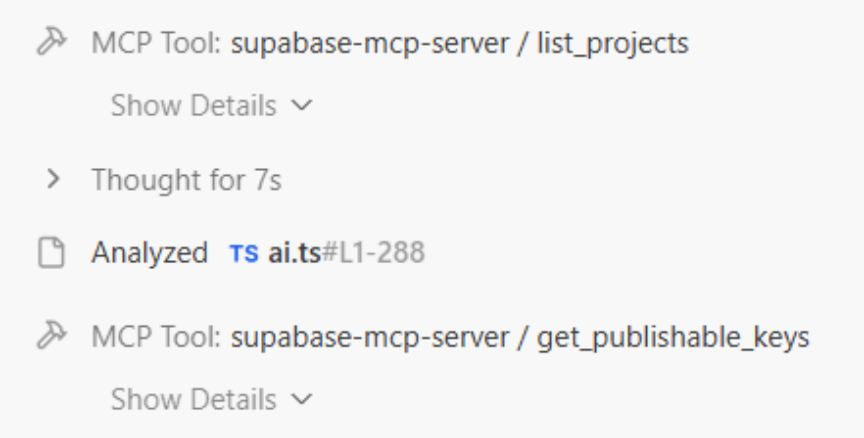
Supabase is now installed and enabled, and it is ready for use.



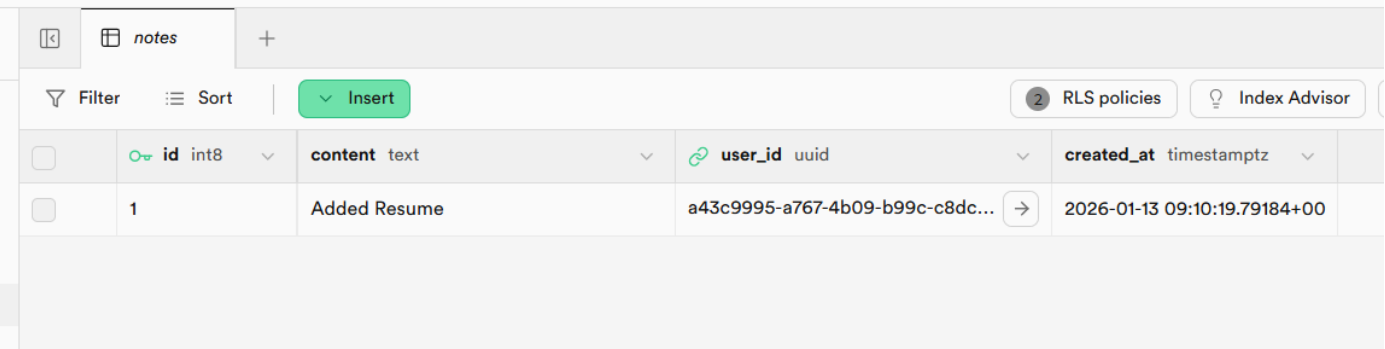
You can write a prompt to integrate **Supabase** based on your specific use case.



In the agent window, you can observe that the agent is successfully connecting to the **MCP tool**.

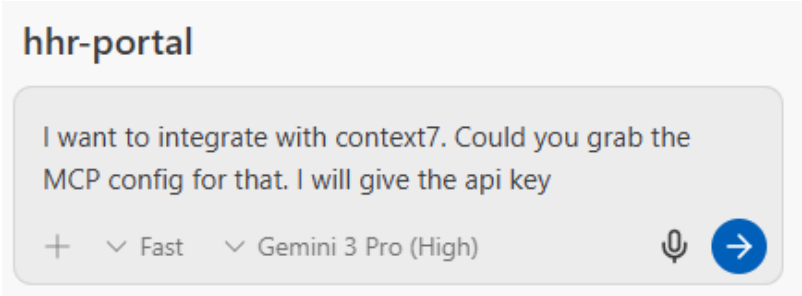


As shown, a table has been created and a record has been successfully inserted.



4.3 Integrating Context7 MCP Server

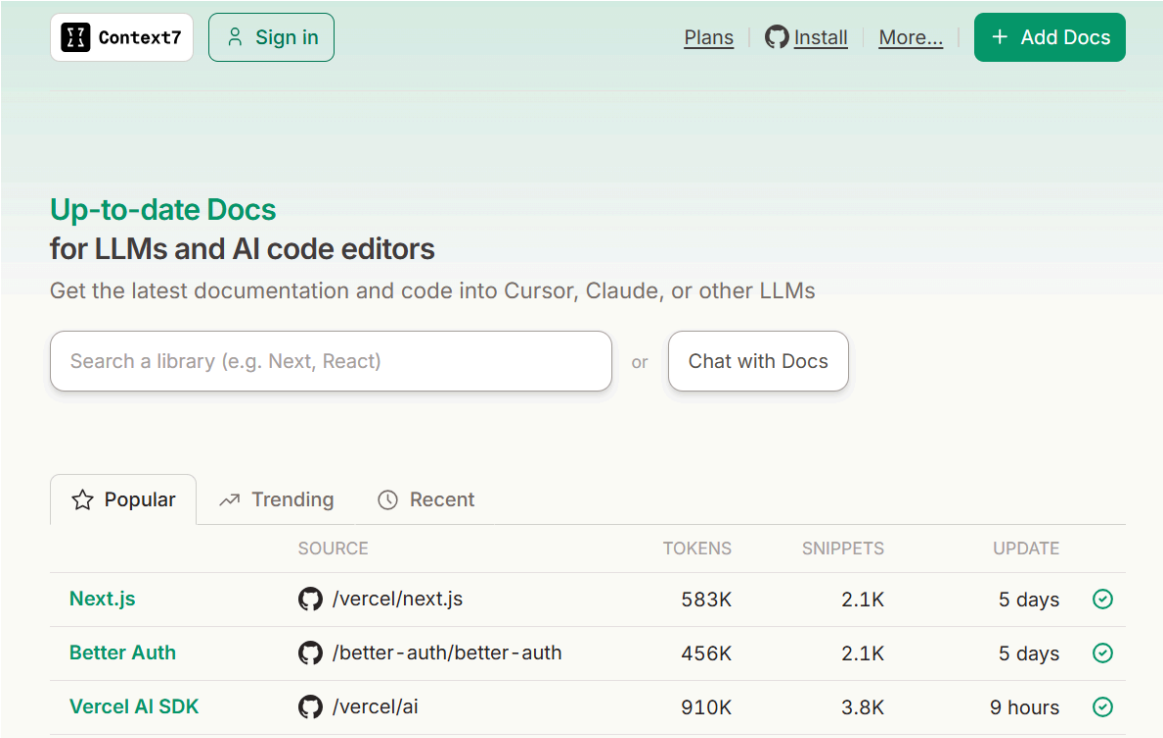
Context7 is not available in the MCP Store, so it must be integrated manually



Kasadara

You will receive the configuration code from the agent and then visit the Context7 website to obtain the API key.

```json
{
  "mcpServers": {
    "context7": {
      "command": "npx",
      "args": [
        "-y",
        "context7"
      ],
      "env": {
        "CONTEXT7_API_KEY": "YOUR_API_KEY_HERE"
      }
    }
  }
}
```



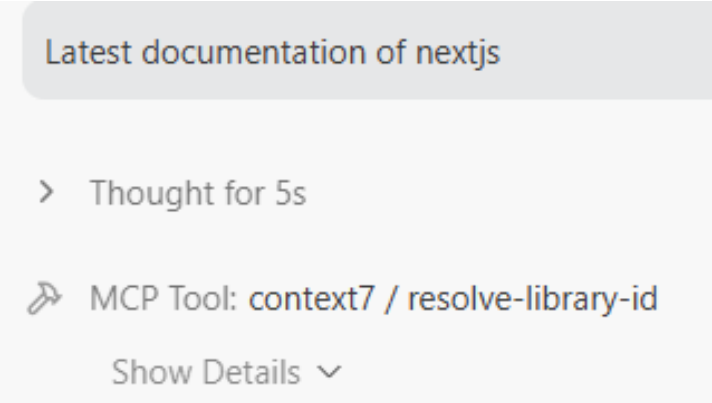
Kasadara

Go to **Manage MCP Servers** → **View Raw Config**, paste the configuration code, and replace the API key placeholder with the actual API key.

```
{
  "mcpServers": {
    "supabase-mcp-server": {
      "command": "npx",
      "args": [
        "-y",
        "@supabase/mcp-server-supabase@latest",
        "--access-token",
        "sbp_e78e7e3956ce2259028dc2d1f96b59cfc9cb98f6"
      ],
      "env": {}
    },
    "context7": {
      "command": "npx",
      "args": [
        "-y",
        "@upstash/context7-mcp"
      ],
      "env": {
        "CONTEXT7_API_KEY": "ctx7sk-eb35db42-18f1-4431-85fe-72a821f301b8"
      }
    }
  }
}
```

After clicking **Refresh** on the **Manage MCP Servers** screen, you can see **Context7** listed as available

Context7 MCP pulls up-to-date, version-specific documentation and code examples straight from the source — and places them directly into your prompt.



5. Parallel Agents

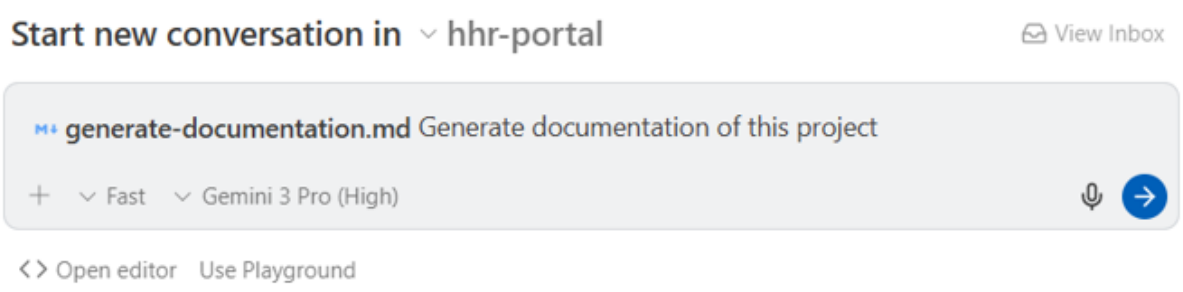
Now, let us see how to run multiple agents in parallel.

Open the **Agent Manager** from the top-right corner in **Google Antigravity**.

In the workspace, click **Add**.

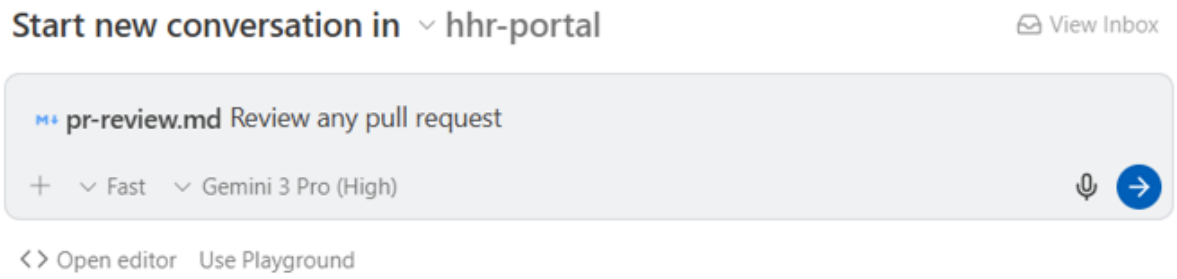
Agent 1:

This agent generates project-wide documentation in **.md (Markdown)** format



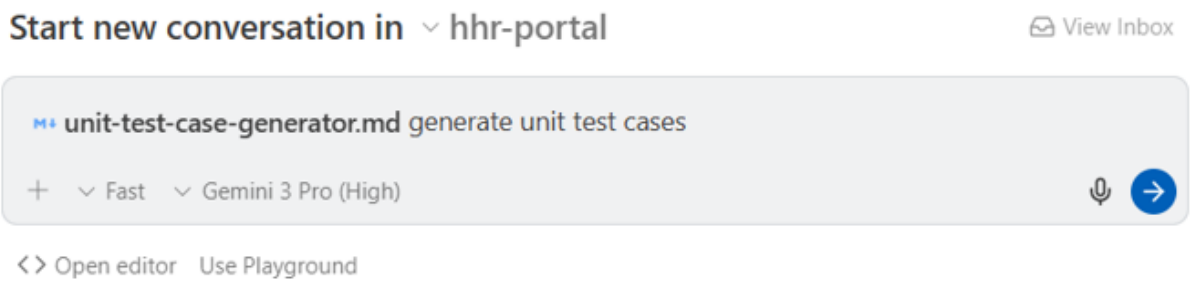
Agent 2:

This agent reviews pull requests



Agent 3:

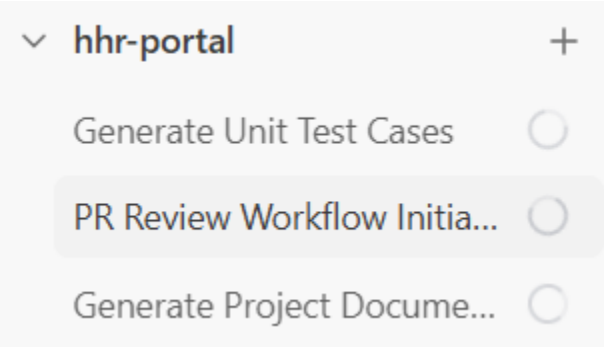
This agent generates unit test cases for the project



- The Agent Manager allows you to create, configure, and manage multiple agents.
- It enables running multiple agents in parallel for different tasks.
- Each agent can be assigned a specific responsibility, such as documentation, code review, or testing

- The Agent Manager helps streamline workflows by coordinating agent execution and monitoring their progress from a single interface

As shown in the screenshot, multiple agents are running in parallel



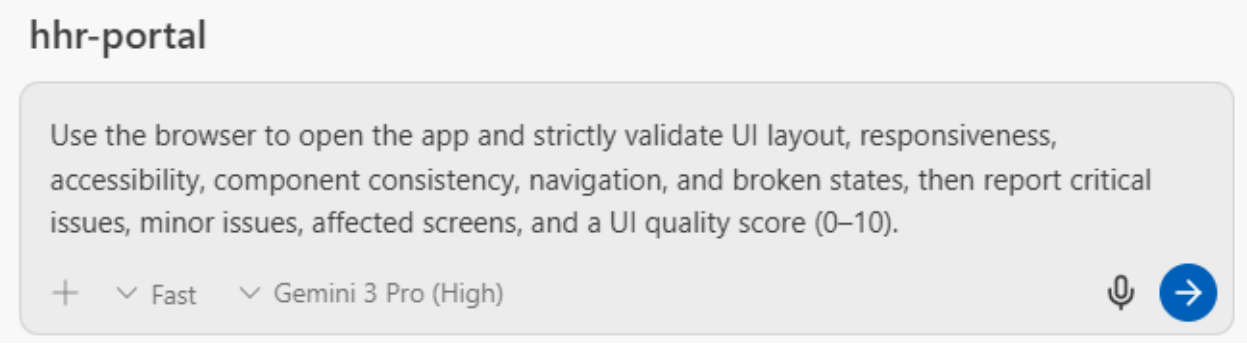
6. Agentic Browser and UI Validation

AI agents can autonomously perform browser-based UI validation

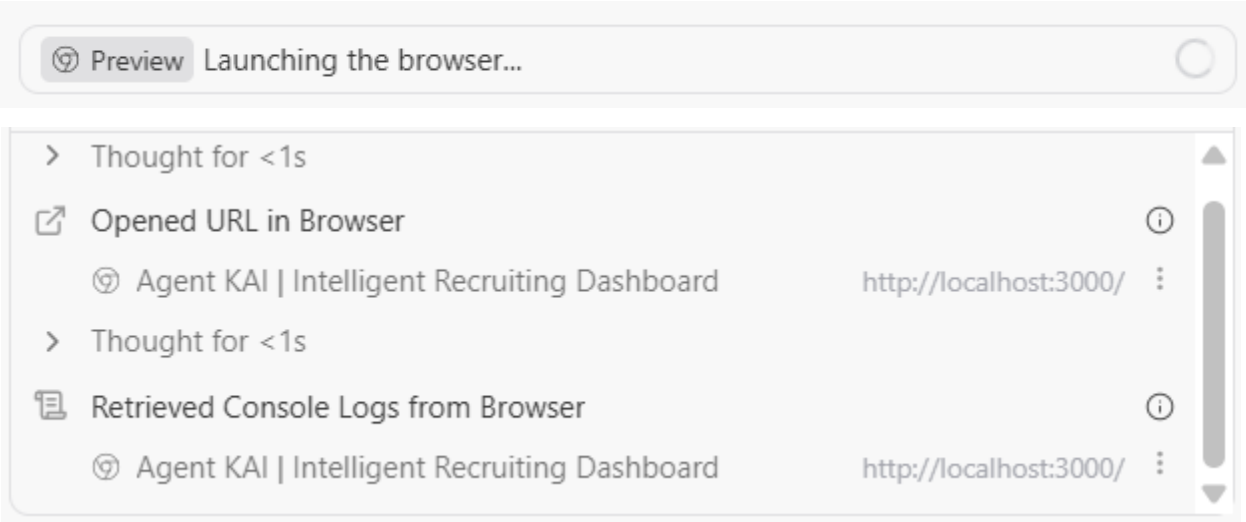
Browser Subagent can:

- Navigate to specified URLs.
- Interact with page elements such as clicking buttons, filling forms, and scrolling.
- Inspect the DOM and read browser console logs.
- Execute JavaScript with configurable security controls

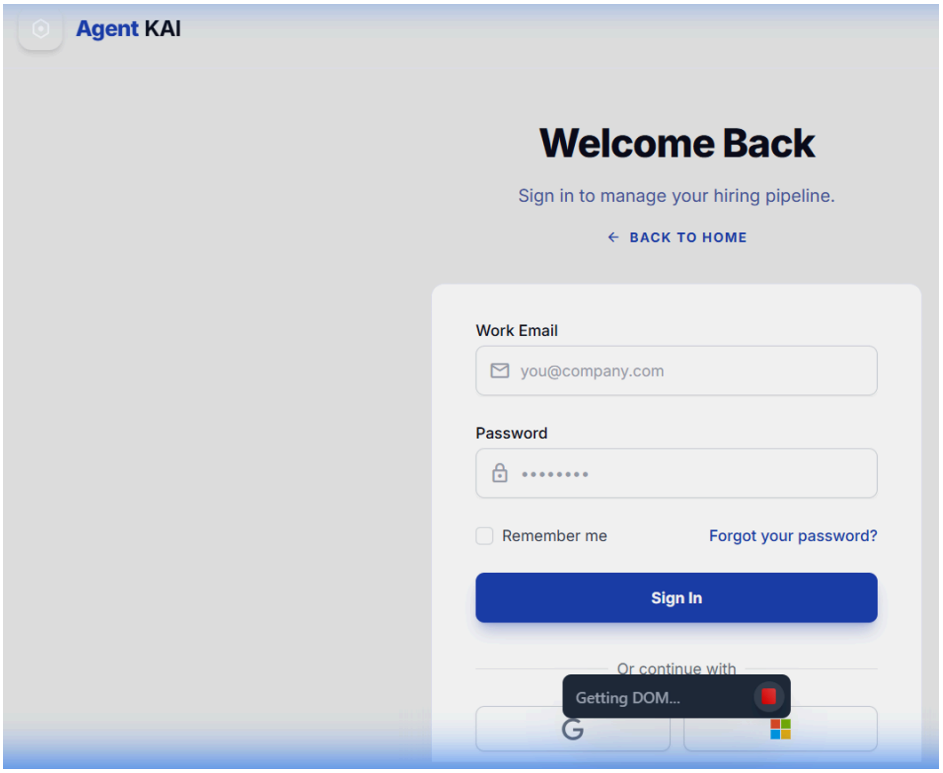
Let us walk through how the Agentic Browser and UI Validation work

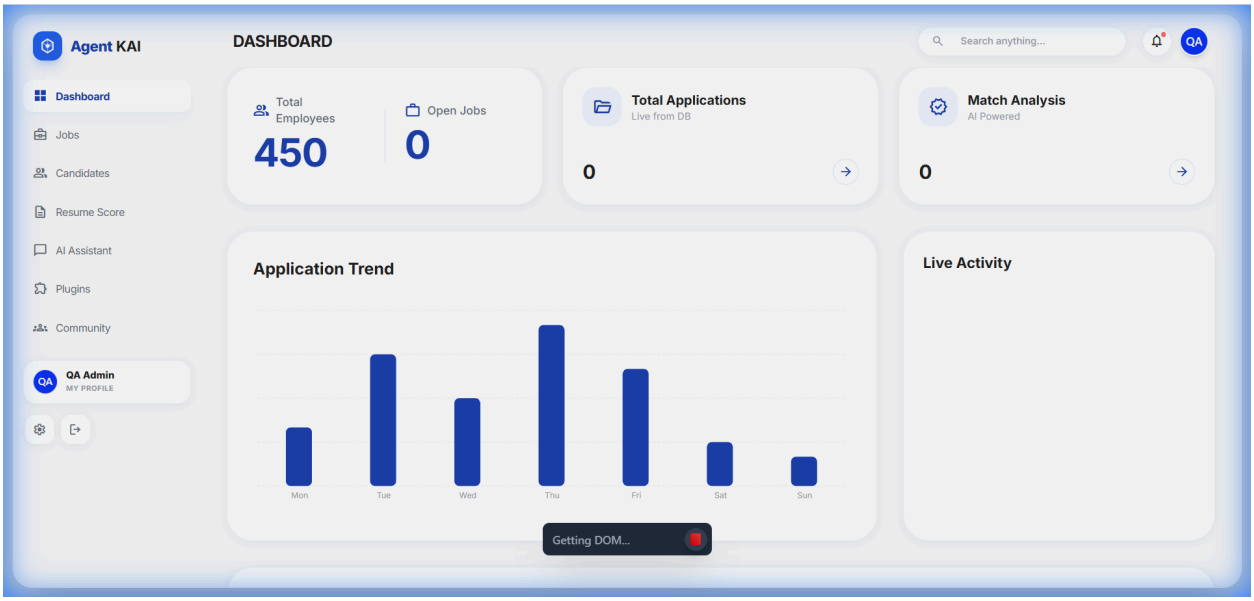


You can see that it automatically opens the browser and performs the testing

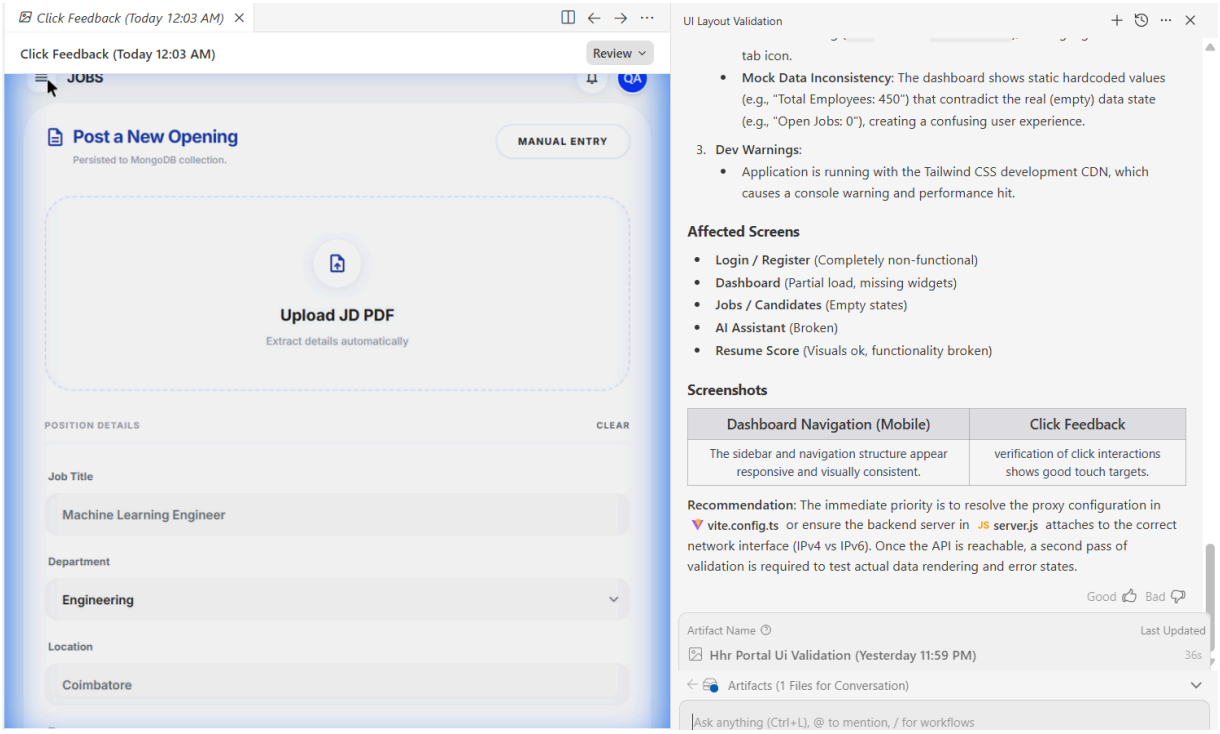


Interactions:





Click Feedback:



Kasadara

UI Validation Report

Overall Quality Score: 3/10 *While the visual design and layout show promise with a modern aesthetic, the application is functionally broken due to critical backend connectivity issues.*

Critical Issues

1. **Backend API Disconnected (Major)**:
   - **Symptoms**: All API calls (Jobs, Candidates, Activities) fail with `500 Internal Server Error` or Proxy Errors (`ECONNREFUSED`).
   - **Root Cause**: The frontend (Vite) cannot connect to the backend server on port 5001. Terminal logs show repeated `http proxy error: ... ECONNREFUSED` when trying to reach `/api/*` endpoints. This renders the application largely unusable.
2. **Broken Authentication**:
   - User registration and login fail completely. The QA agent had to manually

Affected Screens

- **Login / Register** (Completely non-functional)
- **Dashboard** (Partial load, missing widgets)
- **Jobs / Candidates** (Empty states)
- **AI Assistant** (Broken)
- **Resume Score** (Visuals ok, functionality broken)

Screenshots

Dashboard Navigation (Mobile)	Click Feedback
The sidebar and navigation structure appear responsive and visually consistent.	verification of click interactions shows good touch targets.

Recommendation: The immediate priority is to resolve the proxy configuration in vite.config.ts or ensure the backend server in JS server.js attaches to the correct network interface (IPv4 vs IPv6). Once the API is reachable, a second pass of validation is required to test actual data rendering and error states.

Kasadara

7. Conclusion

- This document confirms that **Google Antigravity** functions as a **production-ready agentic development platform**, not just an AI assistant.

- **MCP server integrations** enable agents to interact with live systems, databases, vector stores, and documentation.

- **Context7, Pinecone, and Supabase MCPs** provide accurate documentation, persistent memory, and direct backend operations.

- **Rules, workflows, and parallel agents** ensure controlled, scalable, and repeatable engineering execution.

- **Agentic browser and UI validation** enable real browser-based testing and frontend verification.

- Overall, the implementation demonstrates a **scalable and reliable multi-agent development model** suitable for real-world engineering workflows.