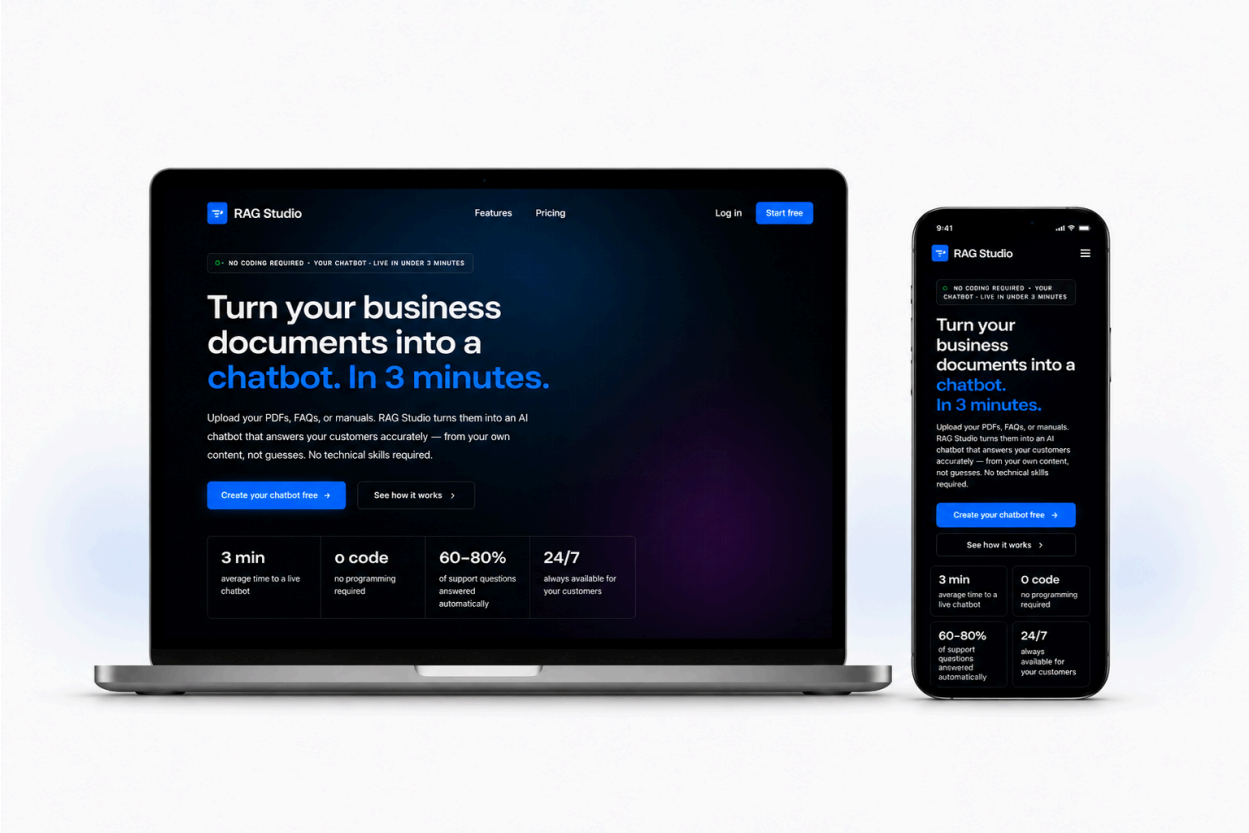
RAG Studio

An AI-powered chatbot platform that automatically transforms business documents into intelligent chatbots for customer support, knowledge retrieval, and website assistance, eliminating the need for manual responses.



Tech Stack

Python • Django • DRF • PostgreSQL • Celery • Redis • ChromaDB • Next.js • React • TypeScript • OpenAI • Stripe • Docker • nginx

The Challenge

Businesses often struggle to provide fast, accurate customer support because valuable information is scattered across documents, manuals, FAQs, and internal knowledge bases. Finding the right answers manually is time-consuming, inconsistent, and difficult to scale as the business grows.

Traditional chatbots rely on predefined responses and require significant setup or technical expertise, making them difficult to maintain and unable to answer questions based on company-specific knowledge. As documentation evolves, keeping chatbot responses accurate becomes an ongoing challenge.

An end-to-end solution was needed that could transform business documents into an intelligent knowledge base, enable users to test AI responses, and deploy a production-ready chatbot to their website with minimal setup and no custom development.

Target Users

- Small and medium-sized businesses (SMBs)
- Customer support and customer success teams
- Internal knowledge management and operations teams
- SaaS companies and digital product businesses
- Organizations managing documentation, manuals, and FAQs
- Teams looking to deploy AI-powered chatbots without custom development

Solution

RAG Studio streamlines the entire AI chatbot creation workflow, from document upload to website deployment. The platform automatically processes business documents, generates vector embeddings, and builds an intelligent knowledge base for retrieval-augmented generation (RAG). Users can test chatbot responses in real time before deploying the chatbot to their website with a single embed code. By eliminating complex AI setup and custom development, RAG Studio enables businesses to launch accurate, context-aware chatbots quickly and at scale.

Key Features

Document Processing: Automatically uploads, parses, chunks, and indexes business documents in the background.

AI-Powered Chat: Generates accurate, context-aware responses using Retrieval-Augmented Generation (RAG).

Semantic Search: Retrieves relevant information using vector embeddings instead of keyword matching.

Project-Based Knowledge Bases: Organize documents into isolated projects with dedicated knowledge repositories.

Real-Time Processing: Monitor document indexing and processing progress through live WebSocket updates.

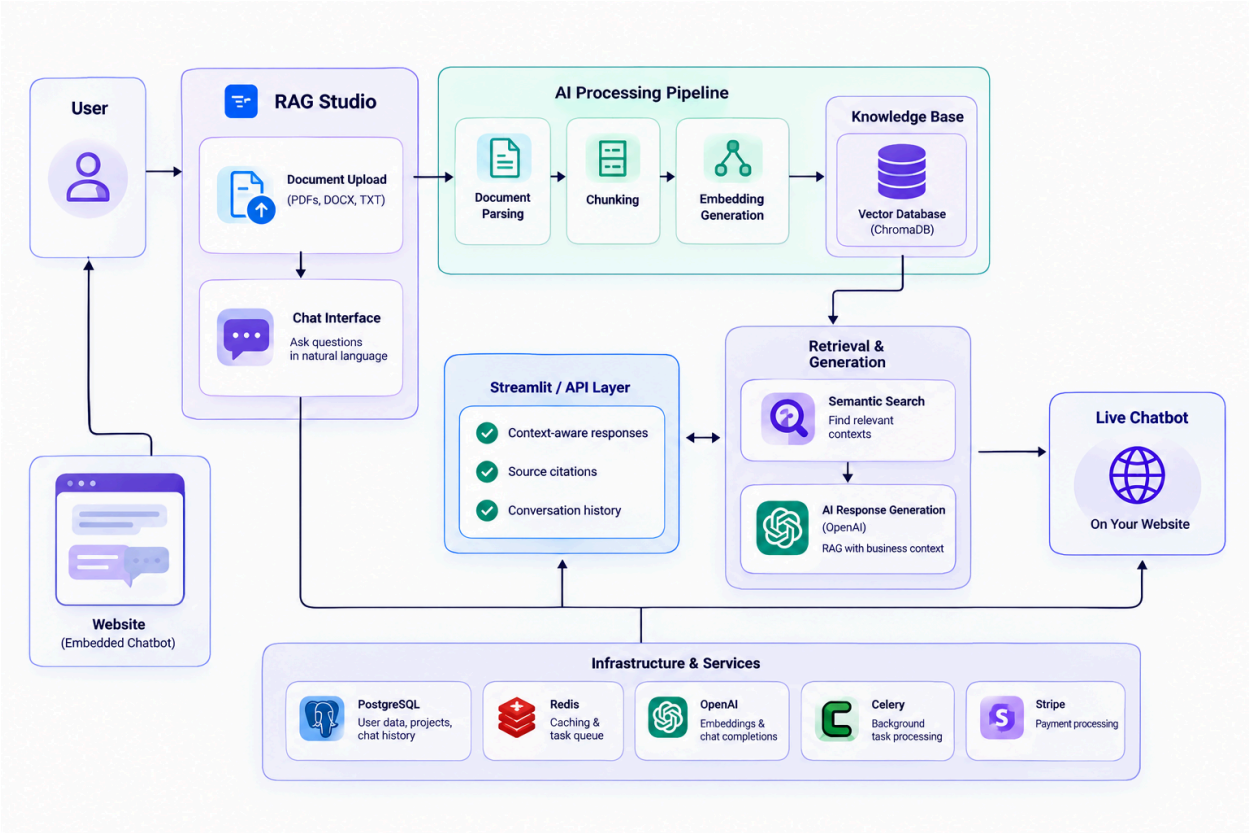
Team Collaboration: Multi-tenant workspaces with role-based access control and team invitations.

Chat History: Maintain persistent conversation threads for continuous, context-aware interactions.

Website Integration: Deploy AI chatbots to any website using a simple embed code with minimal configuration.

Workflow Diagram

Technical Architecture



Challenges & Solutions

Challenge 1: Unstructured Business Documents

Challenge: Businesses stored valuable information across PDFs, manuals, FAQs, and other documents, making it difficult to retrieve accurate answers quickly.

Solution: Built an automated document processing pipeline that parses, chunks, generates embeddings, and indexes documents into a searchable knowledge base.

Challenge 2: Traditional Chatbots Lacked Context

Challenge: Rule-based chatbots could only answer predefined questions and struggled with business-specific knowledge.

Solution: Implemented a Retrieval-Augmented Generation (RAG) pipeline that performs semantic search and supplies relevant document context to the AI before generating responses.

Challenge 3: Slow Document Processing

Challenge: Processing large documents could delay users and negatively impact the application experience.

Solution: Used Celery workers and Redis to process documents asynchronously while providing real-time progress updates through WebSockets.

Challenge 4: Secure Collaboration Across Teams

Challenge: Multiple organizations needed isolated workspaces, secure access, and collaborative project management without exposing data between teams.

Solution: Implemented a multi-tenant architecture with role-based access control, workspace isolation, and team invitations, ensuring secure collaboration while keeping customer data completely separated.

Production & Scalability

Designed using production-grade engineering practices, with a modular frontend-backend architecture that supports reliable deployments, asynchronous document processing, and future scalability. The platform is containerized with Docker, providing consistent environments across development and production while keeping the application easy to maintain and extend.

Containerized Architecture

The application is split into independent frontend and backend services, allowing each layer to be developed, deployed, and maintained separately.

- Next.js 16 frontend
- Django + Django REST Framework backend
- PostgreSQL for relational data
- ChromaDB for vector storage
- Redis for caching and message brokering
- Celery Workers for background processing

Authentication & Authorization

The platform uses NextAuth with JWT-based authentication and implements role-based access control (RBAC) to manage permissions for workspace Owners, Admins, and Members. Multi-tenant workspace isolation ensures each organization's data remains securely separated.

Background Processing

Resource-intensive operations are processed asynchronously using Celery with Redis, allowing the application to remain responsive while processing documents in the background.

Background jobs include:

- Document parsing
- Text chunking
- Embedding generation
- Vector indexing
- Scheduled maintenance tasks

Real-Time Updates

Document processing progress is delivered through WebSockets (Django Channels), enabling users to monitor indexing status and task completion in real time without refreshing the application.

Security

The application incorporates several production-ready security practices, including:

- JWT authentication
- Email verification
- Role-based access control
- Workspace-level data isolation
- Environment-based secrets management
- ORM-based SQL injection protection

Database & Performance

RAG Studio is optimized for fast document retrieval and efficient AI inference using:

- PostgreSQL for application data
- ChromaDB for semantic vector search
- Redis for caching and task coordination
- Optimized document chunking and retrieval pipelines to minimize response latency

Scalability

The current architecture is deployed as a monolithic application with separate frontend and backend services. Its modular design provides a solid foundation for future scaling as usage grows.

Component	Future Scaling Strategy
Frontend	Deploy multiple stateless instances behind a load balancer
Backend	Run multiple Django application instances
Celery Workers	Increase worker processes for higher document-processing throughput
ChromaDB	Migrate to dedicated vector database infrastructure as datasets grow
PostgreSQL	Upgrade to managed database services with backups and replication

Deployment

A CI/CD pipeline automates the build, testing, and deployment workflow, enabling reliable and repeatable releases with minimal manual intervention. The platform is fully containerized with Docker and is cloud-ready, allowing the existing architecture to evolve into a distributed deployment as scalability requirements increase.

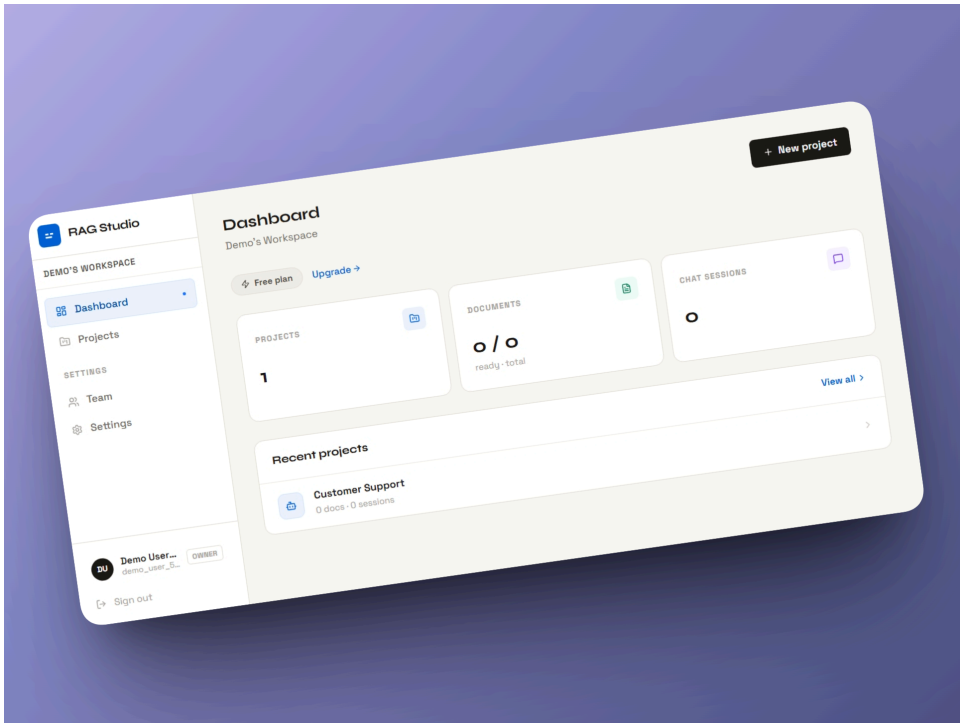
My Responsibilities

As the sole developer, I was responsible for the complete design, development, deployment, and delivery of the platform, from system architecture to production deployment. My responsibilities included:

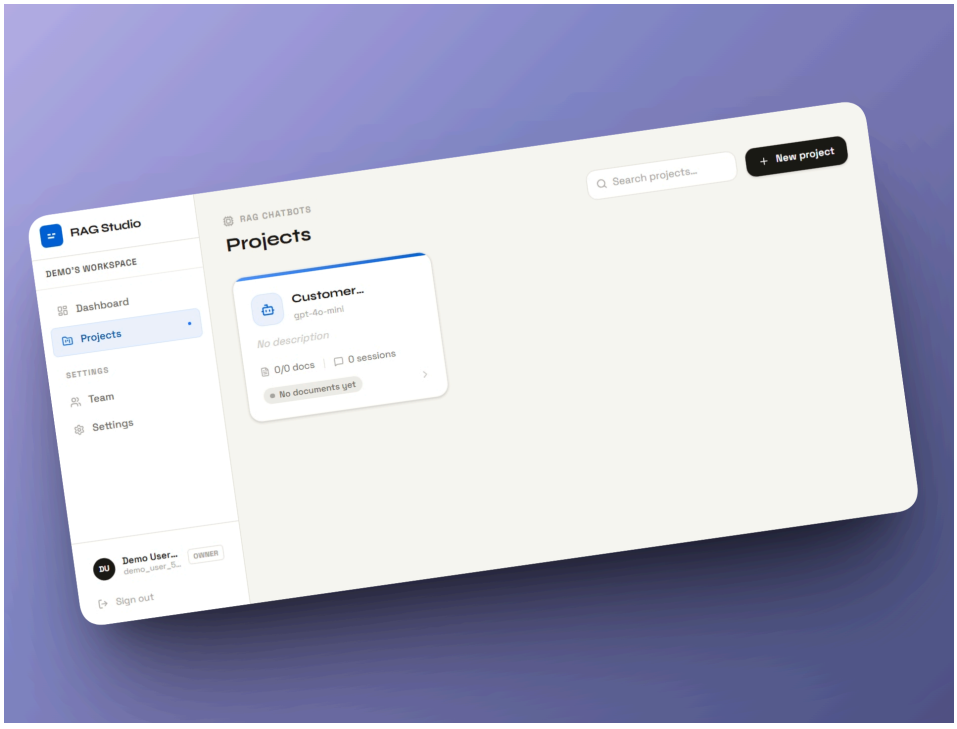
- Designed the overall system architecture, database schema, and multi-tenant workspace model.
- Developed the frontend, backend APIs, and business logic using Next.js, Django, and Python.
- Built the complete RAG pipeline, including document parsing, chunking, embedding generation, semantic search, and AI-powered chat.
- Integrated OpenAI, ChromaDB, Stripe, NextAuth, and real-time WebSocket communication.
- Implemented authentication, role-based access control, team collaboration, billing, and usage tracking.
- Optimized document processing, database queries, vector retrieval, and overall application performance.
- Managed Docker containerization, CI/CD, deployment, environment configuration, testing, and production readiness.

Gallery

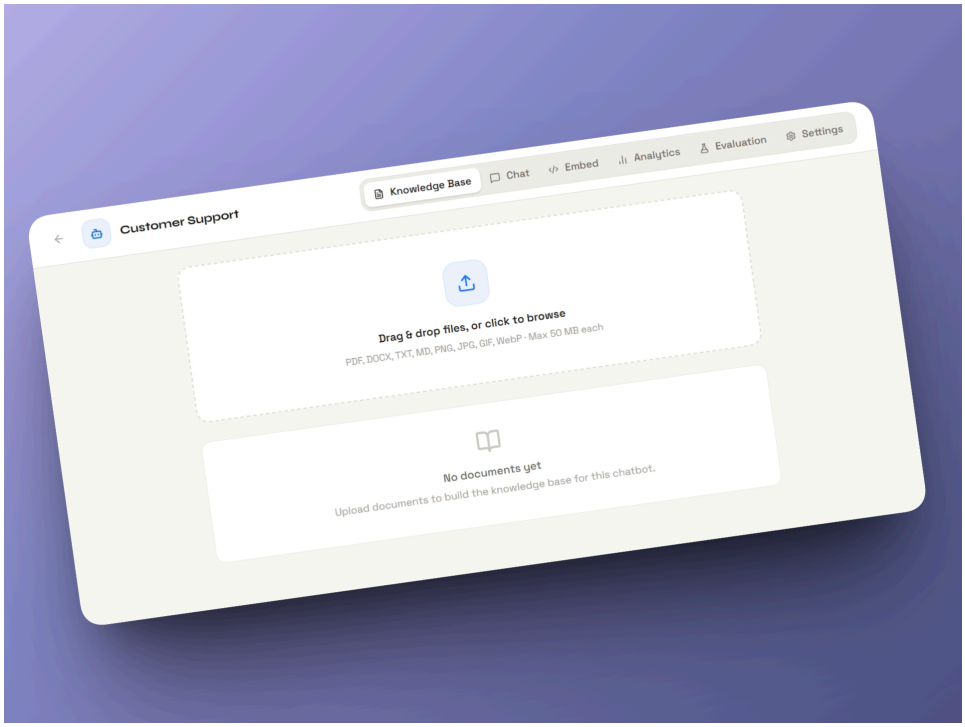
Dashboard



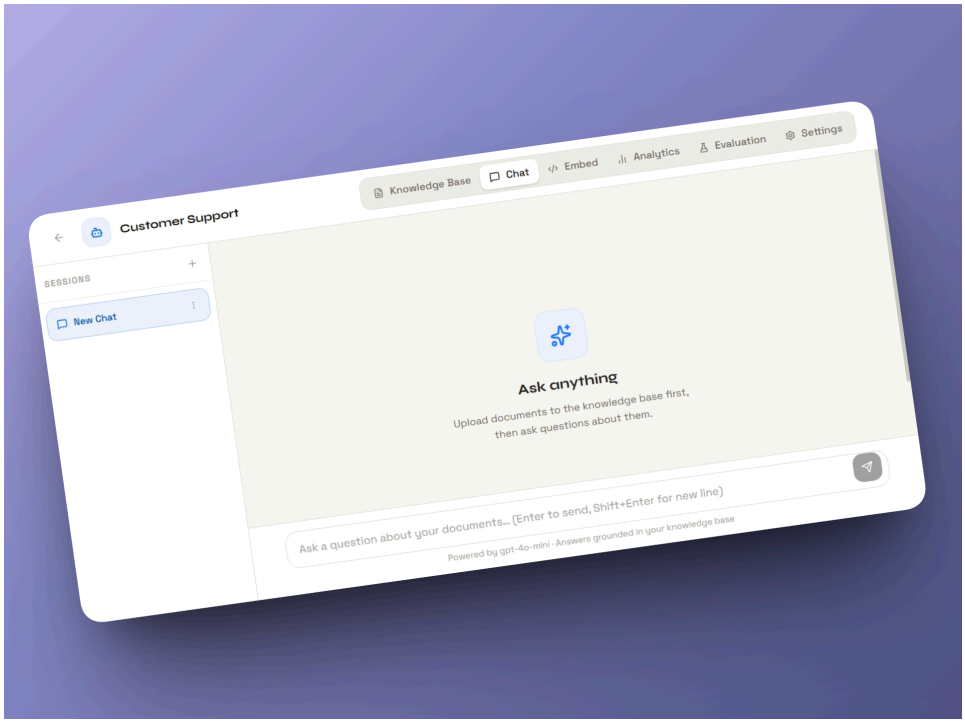
Create Chatbot



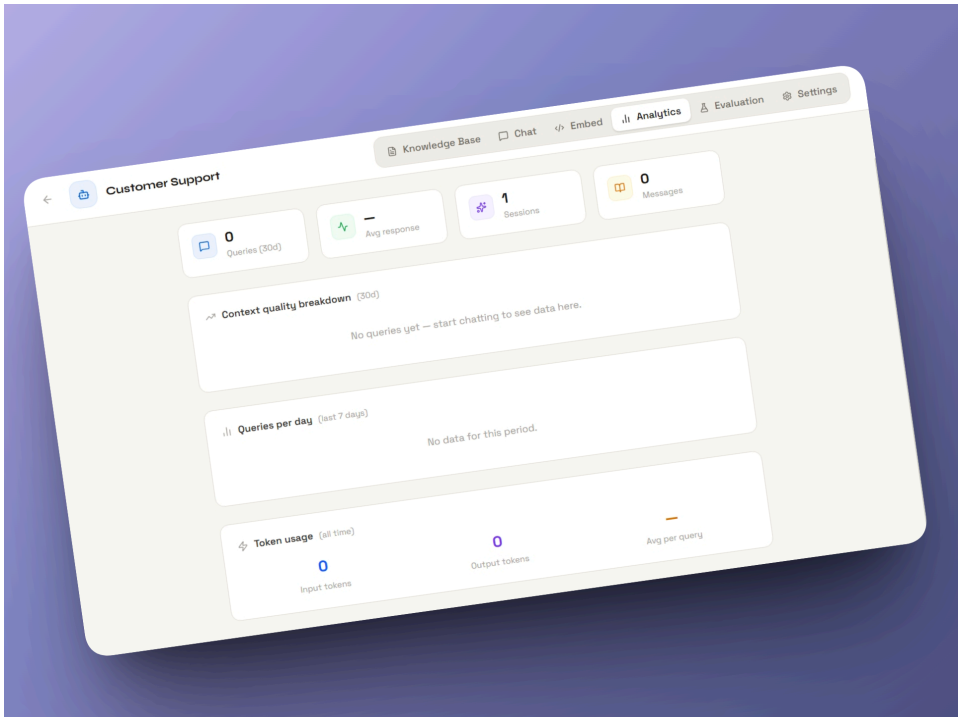
Upload Documents



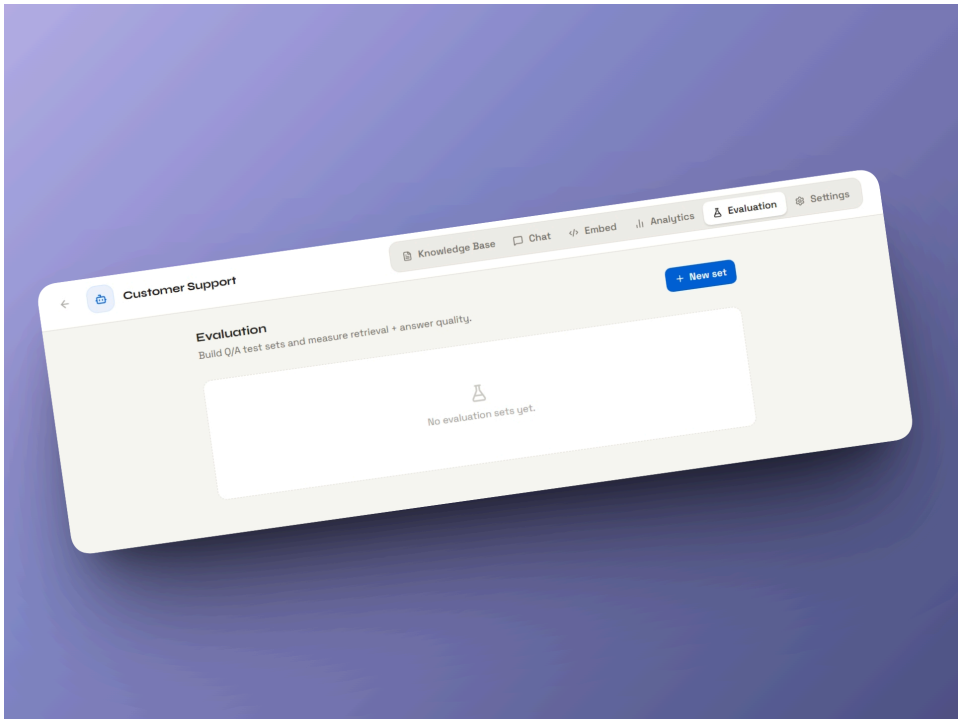
Test Chatbot



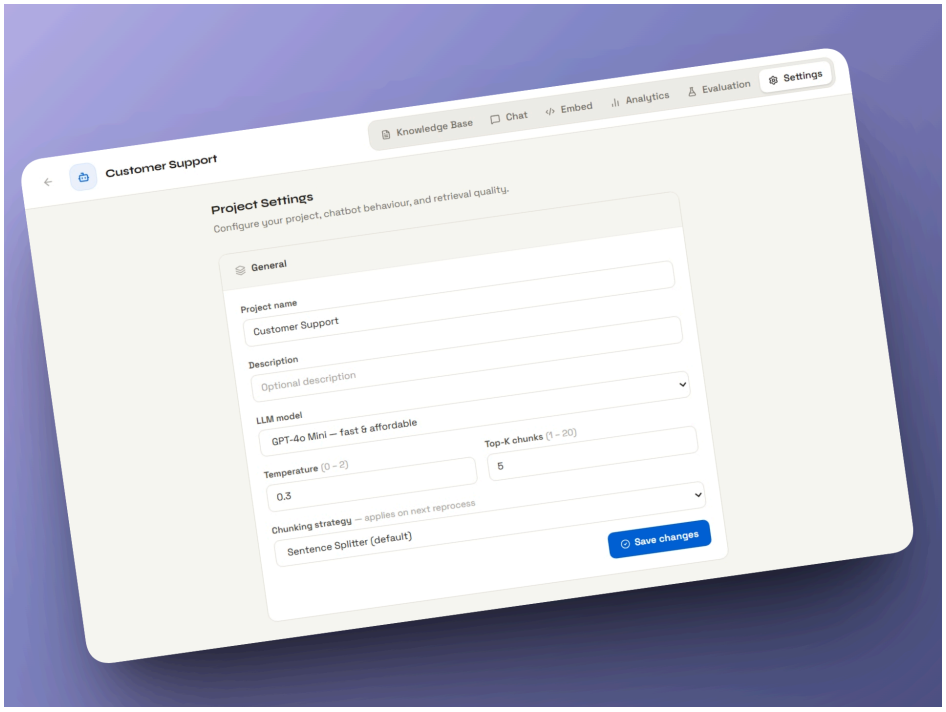
Chatbot Analytics



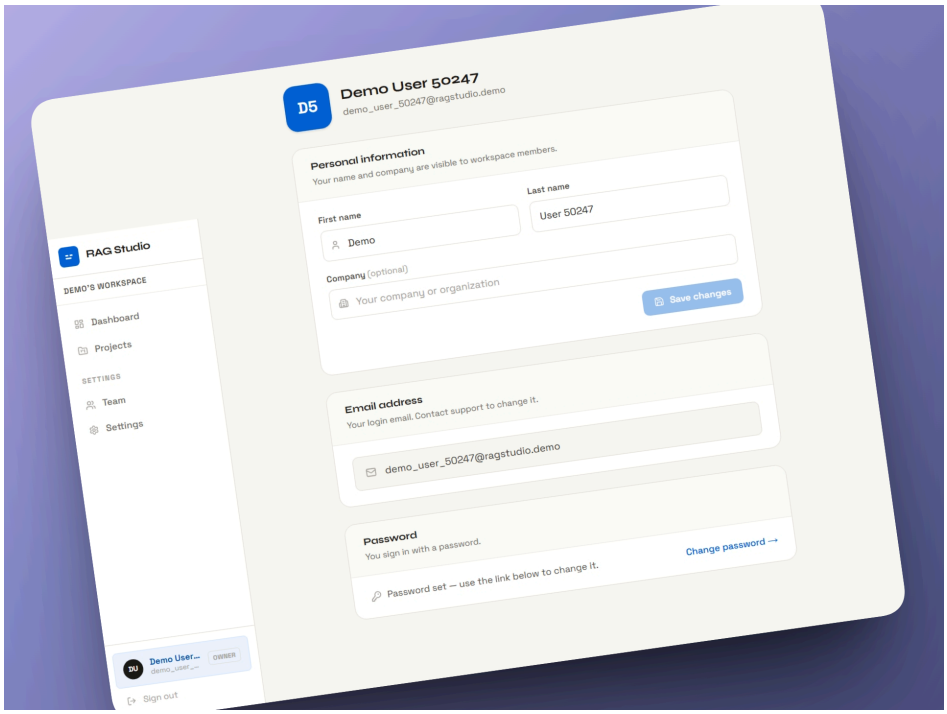
Chatbot AI Unit Tests



Edit Chatbot Settings



User Settings



Project Links

Live demo:
ragstudio.wajahatlabs.com

Github repo:
wajahat-py/ragstudio

Final Takeaway

RAG Studio demonstrates how Retrieval-Augmented Generation (RAG) can transform business knowledge into an intelligent, production-ready AI assistant. By combining automated document processing, semantic search, context-aware AI responses, and seamless website deployment, the platform enables organizations to deliver accurate, scalable, and maintainable AI experiences powered by their own knowledge base.