Reginald – System Architecture & Deployment Whitepaper

Introduction

Reginald is an AI Copilot that answers any question about regulations. It is designed to support compliance analysts and financial crime teams by providing instant, AI-driven responses to regulatory queries. The system leverages a modern full-stack architecture with a secure backend, responsive frontend, and integration with external AI services.

System Overview

At its core, Reginald connects users with an AI chat experience. Analysts log in through the frontend, interact with the dashboard and chat UI, and receive streaming responses directly from the integrated AI model. The backend securely manages authentication, request routing, and communication with external services. The architecture consists of: - Frontend (React + Vite + Tailwind + ShadCN UI) - Backend (FastAPI, Python) - Shared Modules for consistency across services - External AI API (Gemini) - Optional Database Layer (Future) for persistence

Frontend

Technology Stack: React, Vite, Tailwind v4, ShadCN UI Primary Interfaces: - Login Page – secure user access - Dashboard – central workspace - Chat Interface – main AI interaction point Streaming Support: The frontend consumes Server-Sent Events (SSE) from the backend, displaying AI responses progressively word by word for a conversational experience.

Backend

Technology Stack: FastAPI (Python), Uvicorn/ASGI runtime Responsibilities: - Manage authentication and session tokens (JWT) - Provide REST API endpoints (/auth/login, /chat/send) - Deliver streaming AI responses via /chat/stream (SSE) - Proxy securely between frontend and Gemini AI API Scalability: Containerized backend can be deployed on AWS ECS, GKE, or Azure AKS for horizontal scaling.

Shared Modules

Shared resources ensure consistency across frontend and backend, avoiding mismatches. - Types (user session types, API response shapes) - Configs (environment variables, API keys, constants) - Utilities (helper functions reused across layers)

External Services – Gemini API

Role: Provides AI inference and returns natural-language answers. Integration: The backend acts as a secure proxy, never exposing API keys to the frontend. Streaming: Supports real-time delivery of partial responses for improved user experience.

Database Layer (Future)

Though not yet implemented, the system architecture anticipates persistent storage. This will support: - Chat History – enabling analysts to revisit conversations - Audit Logs – providing traceability for compliance reporting - Case Tracking – managing compliance investigations end-to-end

Deployment Overview

Deployment separates responsibilities across cloud services: Frontend Deployment: - Hosted on Vercel or Netlify - Delivered as static assets via CDN - Benefits from edge caching and HTTPS by default Backend Deployment: - Dockerized FastAPI application - Runs on container platforms such as AWS ECS, GKE, or Azure AKS - Scales horizontally with orchestration tools - Secured with HTTPS and environment-based secrets External Services: - Gemini API, accessed via secure proxy - API keys and credentials stored in vaults or secret stores CI/CD Pipeline: - GitHub Actions for automated builds, tests, and deployments - Separate environments (dev, staging, production)

Security & Reliability

Authentication: Handled with JWT or session tokens Data Protection: HTTPS for all communication; no direct AI service exposure Secrets Management: Credentials stored securely in secret vaults Performance: SSE streaming minimizes latency and improves perceived response times Reliability: Fault tolerance and retry mechanisms in backend for API communication

Workflow Example

1. Analyst opens the app → Login through frontend 2. Backend authenticates credentials → issues JWT 3. Analyst accesses dashboard and opens chat 4. Analyst submits query → backend routes it to Gemini API 5. Gemini processes request → streams response chunks 6. Backend relays streamed response to frontend via SSE 7. Frontend renders word-by-word output until complete 8. (Future) Data stored in database for history and audit logs

Future Enhancements

- Persistent storage (Postgres/MongoDB) for compliance records - Advanced knowledge base integration (policy documents, regulatory references) - Role-based access control (Admin vs Analyst) - Audit reporting features - Multilingual support for global compliance teams - Additional AI model integrations or fine-tuning

Conclusion

The Reginald System Architecture provides a scalable, secure, and extensible platform for AI-driven compliance assistance. By combining a modern frontend, efficient backend, and external AI capabilities, Reginald supports compliance professionals in addressing complex regulatory questions quickly and reliably. Future extensions will strengthen its persistence, audit, and workflow management features, making it an indispensable tool for financial crime and compliance operations.