Full Stack Architecture Documentation for LLM Applications

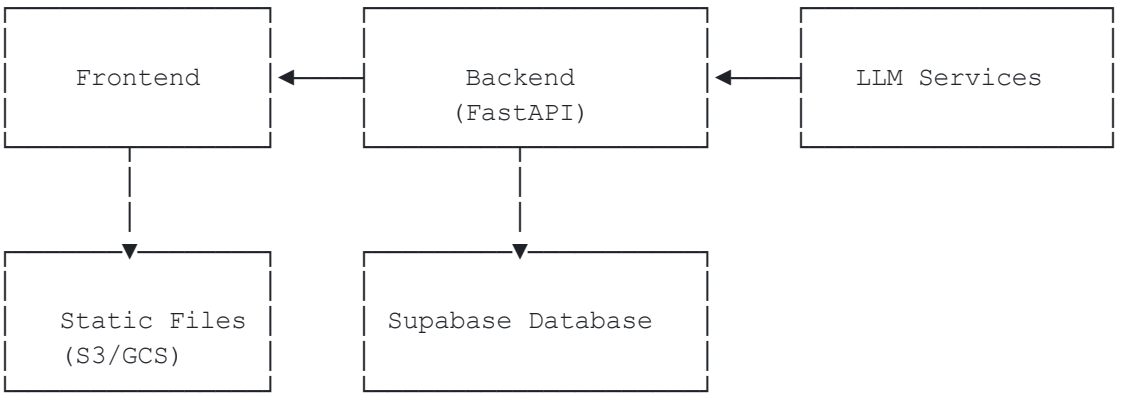
Table of Contents

Introduction

This documentation outlines the architecture for developing a full-stack application that incorporates Large Language Models (LLMs), using AWS/GCP cloud services, Supabase for database management, and FastAPI for the backend framework. The architecture is designed to be scalable, secure, and maintainable while optimizing for performance and cost efficiency.

Overall Architecture

High-Level Architecture Diagram



Architecture Visualization Tools

For creating and maintaining architecture diagrams, we recommend the following tools:

1. Excalidraw
    - Lightweight, browser-based diagramming tool
    - Collaborative features for team diagramming sessions
    - Simple, sketch-like aesthetics that are perfect for ideation
    - Export options to PNG, SVG, and JSON formats
    - Best for: Quick architecture sketches, whiteboarding sessions, and initial design concepts
2. draw.io (diagrams.net)
    - Comprehensive diagramming tool with extensive shape libraries
    - Integration with version control systems (GitHub, GitLab)
    - Support for cloud storage platforms (Google Drive, OneDrive)
    - Export to multiple formats (PNG, SVG, PDF, HTML)
    - Best for: Detailed architecture diagrams, network diagrams, and flow charts
3. Best Practices for Architecture Diagramming
    - Use consistent notation (preferably C4 model or UML)
    - Create diagrams at multiple levels of abstraction (system context, container, component)
    - Include clear labels and descriptions
    - Version control your diagrams alongside code
    - Update diagrams as architecture evolves

PART 1: Data Science & LLM Engineering

LLM Service Architecture

Components

1. Model Management
    - Model selection and versioning
    - Model deployment strategies (hosted API, self-hosted)
    - Model evaluation and monitoring
    - A/B testing framework
2. Prompt Engineering

    - Prompt template management
    - Prompt versioning and optimization
    - System prompts and user prompt handling
    - Prompt validation and sanitization
3. Inference Pipeline
    - Request preprocessing
    - Token management and optimization
    - Response post-processing
    - Streaming support
4. LLM Orchestration
    - Model routing based on task complexity
    - Fallback mechanisms
    - Chaining and multi-step reasoning
    - Tool usage integration

Integration Methods

1. Direct API Integration
    - Connecting to commercial LLM providers (OpenAI, Anthropic, etc.)
    - API key management and rotation
    - Rate limiting and quota management
    - Vendor-specific optimizations
2. Self-hosted Models
    - Model deployment on AWS/GCP resources
    - Containerized model serving (Docker, Kubernetes)
    - Quantization and optimization
    - Hardware acceleration (GPU/TPU)
3. Serverless Functions
    - AWS Lambda or GCP Cloud Functions for LLM processing
    - Request batching and throttling
    - Cold start optimization
    - Memory and timeout configuration

Best Practices

1. Prompt Engineering

```
// Example template structure
{
  "system_prompt": "You are a helpful assistant that...",
  "user_template": "USER: {{user_input}}\nCONTEXT: {{context}}",
  "assistant_template": "ASSISTANT: ",
  "metadata": {
    "version": "1.2",
    "author": "AI Team",
```

```
    "use_case": "Customer Support"
  }
}
```

- Store prompts as versioned templates
- Implement systematic prompt testing
- Document prompt patterns and their effectiveness
- Use templating systems for dynamic prompt generation

2. Model Management
   - Track model performance metrics (accuracy, latency, cost)
   - Implement canary deployments for new models
   - Create model registries with metadata
   - Develop model documentation standards
3. Response Processing
   - Implement content filtering and safety measures
   - Use structured outputs (JSON mode) when appropriate
   - Process and validate responses before returning to users
   - Handle hallucinations and incorrect outputs
4. Caching Strategy
   - Cache common LLM responses to reduce latency and costs
   - Implement intelligent cache invalidation
   - Use vector similarity for approximate matching
   - Store response metadata with cache entries
5. Cost and Performance Optimization
   - Implement token counting and budget controls
   - Use appropriate model sizes based on task complexity
   - Batch similar requests when possible
   - Monitor token usage trends and optimize high-usage patterns

LLM Infrastructure

AWS Infrastructure for LLM

1. Compute Options
   - SageMaker for model hosting
   - EC2 with GPU instances for custom deployments
   - Lambda for serverless inference
   - ECS/EKS for containerized model serving
2. Supporting Services
   - ElastiCache for response caching
   - SQS for request queueing

    - CloudWatch for monitoring and logging
    - S3 for model artifact storage

GCP Infrastructure for LLM

1. Compute Options
    - Vertex AI for model hosting and management
    - Compute Engine with GPU/TPU instances
    - Cloud Run for containerized model serving
    - Cloud Functions for serverless inference
2. Supporting Services
    - Memorystore for response caching
    - Pub/Sub for request queueing
    - Cloud Monitoring for performance tracking
    - Cloud Storage for model artifact storage

Vector Database Integration

1. Options
    - Pinecone for dedicated vector search
    - PostgreSQL with pgvector extension (via Supabase)
    - Milvus or Weaviate for open-source vector DB
    - Redis with vector search capabilities
2. Architecture Patterns
    - Embeddings generation pipeline
    - Vector indexing and retrieval service
    - Hybrid search (keyword + semantic)
    - Document chunking strategies

LLM-Specific Monitoring

1. Key Metrics
    - Token usage by model
    - Response latency
    - Error rates and types
    - User satisfaction metrics
    - Hallucination detection rates
2. Logging Requirements
    - Prompt templates used (non-PII)
    - Model versions
    - Token counts (input/output)
    - Processing times

    - Correlation IDs for request tracing
3. Dashboard Components
    - Token usage trends
    - Cost projections
    - Model performance comparisons
    - User feedback aggregation
    - Error clustering and analysis