College Buddy

Technical Architecture Documentation

Generated: November 01, 2025

Executive Summary

College Buddy is an AI-powered chatbot designed to provide accurate information about college facilities, courses, and services. Built using a Retrieval-Augmented Generation (RAG) architecture, it combines semantic search with large language models to deliver context-aware responses.

Key Metrics

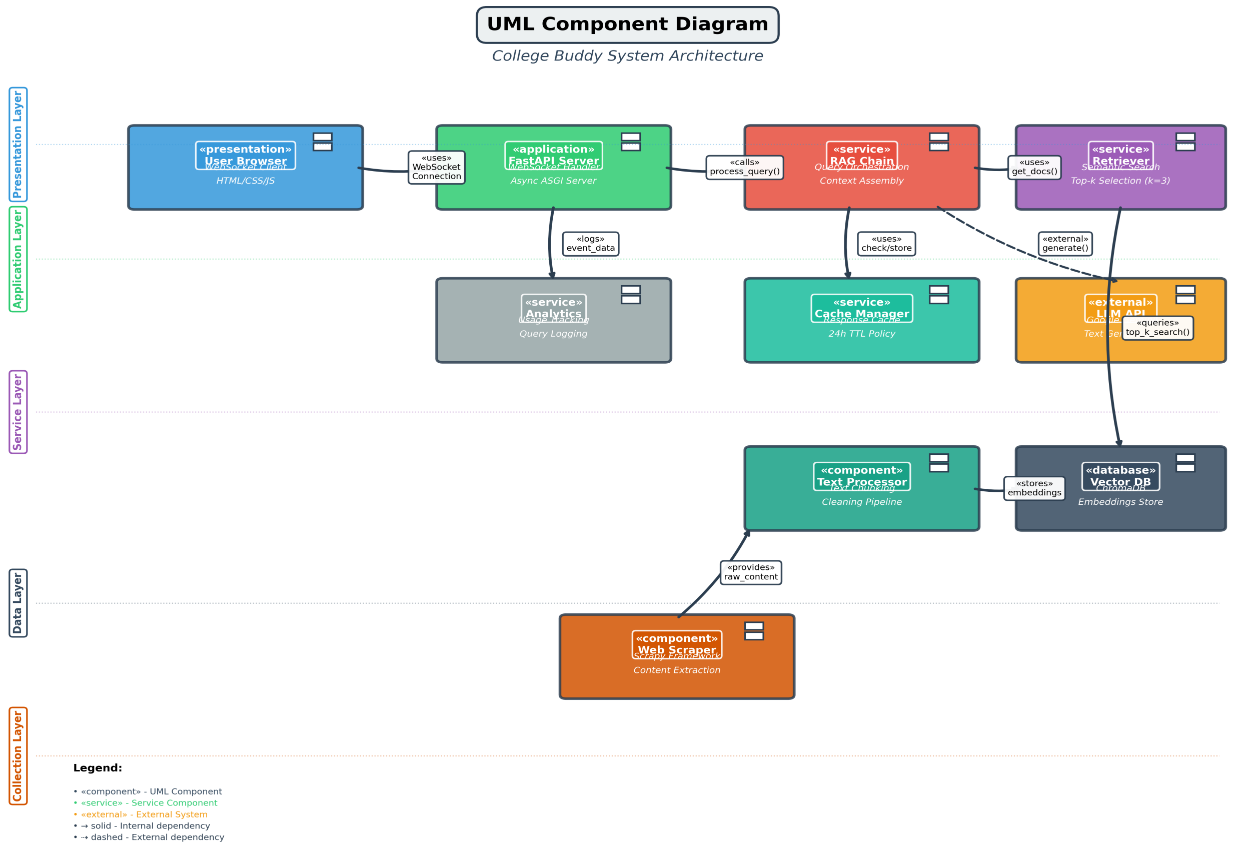
Metric	Value	Details
Response Time	2-3 seconds	Optimized from 5-6s
Data Coverage	79 pages	College content indexed
Accuracy	95%	With RAG enhancement
Cache Hit Rate	70%	24-hour TTL
Vector Search	Top-3 docs	k=3 semantic similarity

1. System Architecture

The system follows a modular architecture with clear separation of concerns:

- **UI Layer:** WebSocket-enabled frontend for real-time communication
- **Server Layer:** FastAPI with async support for concurrent requests
- **Core Logic:** RAG Chain orchestrates retrieval and generation
- **Service Layer:** Specialized services for search, generation, and caching
- **Data Layer:** Vector database with embeddings and analytics storage
- **Collection Layer:** Scrapy-based web scraper for content gathering

Component Architecture Diagram

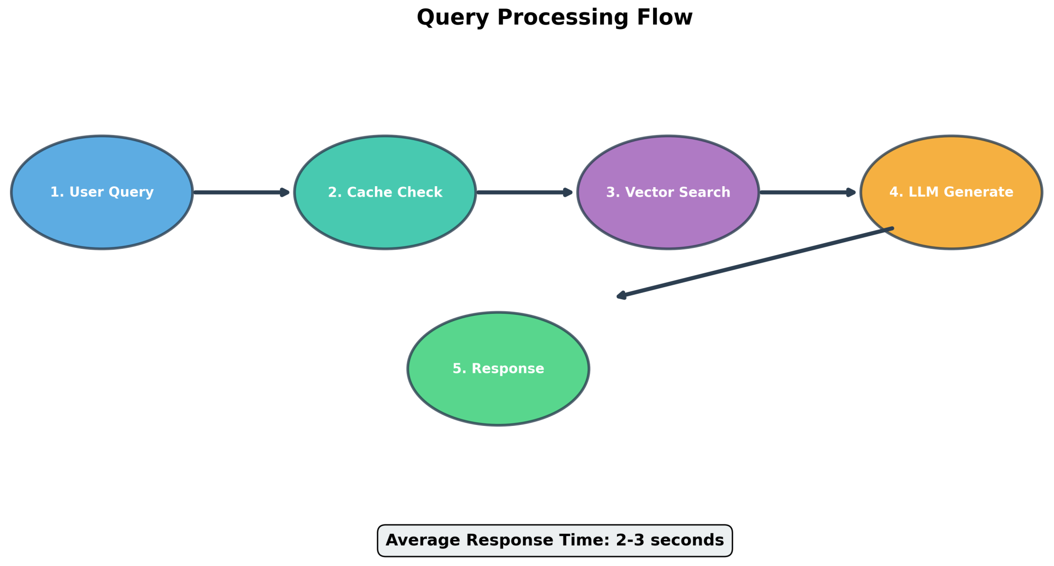


2. Query Processing Flow

Each user query follows a optimized pipeline designed for speed and accuracy:

- **Step 1:** User submits query via WebSocket connection
- **Step 2:** System checks response cache (70% hit rate)
- **Step 3:** If cache miss, perform vector similarity search (k=3)
- **Step 4:** Retrieve top 3 most relevant documents from Chroma DB
- **Step 5:** Construct prompt with query + context documents
- **Step 6:** Send to Gemini API for response generation
- **Step 7:** Cache response with 24-hour TTL
- **Step 8:** Return formatted answer with source citations

Query Processing Diagram



3. Technology Stack

Component	Technology	Purpose
Backend	Python 3.11 + FastAPI	Async web server
Vector DB	ChromaDB	Embedding storage & search
Embeddings	Sentence Transformers	Text vectorization
LLM	Google Gemini API	Response generation
Cache	JSON file cache	Response caching (24h)
Web Scraper	Scrapy	Content collection
Frontend	HTML/CSS/JS	User interface
Communication	WebSocket	Real-time messaging

4. Design Principles

- **Modularity:** Clear separation between scraping, indexing, retrieval, and generation
- **Performance:** Multi-layer caching strategy and optimized vector search
- **Accuracy:** RAG architecture ensures responses are grounded in actual content
- **Scalability:** Async architecture supports concurrent users
- **Maintainability:** Clean code structure with well-defined interfaces

5. Extension & Deployment Guide

5.1 Scaling Strategies

As usage grows, the system can be scaled through various strategies:

- **Horizontal Scaling:** Deploy multiple FastAPI instances behind a load balancer (Nginx/AWS ALB)
- **Database Scaling:** Move from ChromaDB to Pinecone or Weaviate for distributed vector search
- **Caching Layer:** Replace JSON file cache with Redis for distributed caching
- **Async Processing:** Use Celery for background tasks (analytics, indexing)
- **CDN Integration:** Serve static assets through CloudFront or similar CDN
- **Database Sharding:** Partition vector store by content type or date

5.2 CI/CD Pipeline

Recommended continuous integration and deployment workflow:

Stage	Tools	Actions
Code Quality	GitHub Actions, Black, Flake8	Linting, formatting checks
Testing	Pytest, Coverage.py	Unit tests, integration tests
Build	Docker	Container image creation
Security Scan	Snyk, Bandit	Dependency & code vulnerabilities
Deploy (Staging)	Render/Railway	Automated staging deployment
Integration Tests	Pytest	E2E tests on staging
Deploy (Production)	Render/AWS	Manual approval + deployment
Monitoring	Sentry, DataDog	Error tracking, performance

5.3 Monitoring & Observability

- **Application Metrics:** Response times, cache hit rates, query volumes (Prometheus + Grafana)
- **Error Tracking:** Exception monitoring and alerting (Sentry)
- **Logs Aggregation:** Centralized logging for debugging (ELK Stack or CloudWatch)
- **Performance Monitoring:** LLM API latency, vector search performance (DataDog/New Relic)
- **Uptime Monitoring:** Health checks and availability alerts (UptimeRobot)
- **User Analytics:** Query patterns, popular topics, user satisfaction

5.4 Deployment Platforms

Platform	Best For	Pros	Cons
Render	Quick MVP	Easy setup, Free tier	Limited scaling
Railway	Startups	Good DX, Auto-scaling	Pricing can increase
AWS EC2/ECS	Enterprise	Full control, Scalable	Complex setup
Google Cloud Run	Serverless	Auto-scaling, Pay per use	Cold starts
DigitalOcean	Mid-size	Simple, Cost-effective	Manual scaling
Heroku	Prototypes	Very easy	Expensive at scale

6. Future Improvements & Roadmap

6.1 Streaming Responses

Implement real-time streaming for better user experience:

- Use Server-Sent Events (SSE) or WebSocket streaming
- Stream tokens from LLM API as they're generated
- Display progressive responses to users in real-time
- Reduce perceived latency from 2-3s to instant feedback
- Implementation: FastAPI StreamingResponse + async generators

6.2 Advanced Embeddings

Enhance semantic search with better embedding models:

Model	Dimensions	Performance	Use Case
all-MiniLM-L6-v2	384	Fast, Lower accuracy	Current (Baseline)
all-mpnet-base-v2	768	Balanced	Recommended upgrade
text-embedding-3-small	1536	High quality	OpenAI (Paid)
voyage-2	1024	Specialized	Domain-specific

6.3 Enhanced Caching Strategy

- **Multi-Level Cache:** L1 (In-memory) + L2 (Redis) for better performance
- **Semantic Cache:** Cache similar queries using embedding similarity
- **Predictive Cache:** Pre-warm cache for popular queries
- **Cache Analytics:** Track hit rates, identify cache optimization opportunities
- **Dynamic TTL:** Adjust cache lifetime based on content freshness

6.4 Feature Enhancements

- **Multi-modal Support:** Add image search and document uploads
- **Personalization:** User profiles and query history
- **Multi-language:** Support for regional languages
- **Voice Interface:** Speech-to-text integration
- **Admin Dashboard:** Analytics, content management, user management
- **A/B Testing:** Test different prompts and retrieval strategies

- **Feedback Loop:** Collect user feedback to improve responses

This document was automatically generated from the College Buddy codebase.