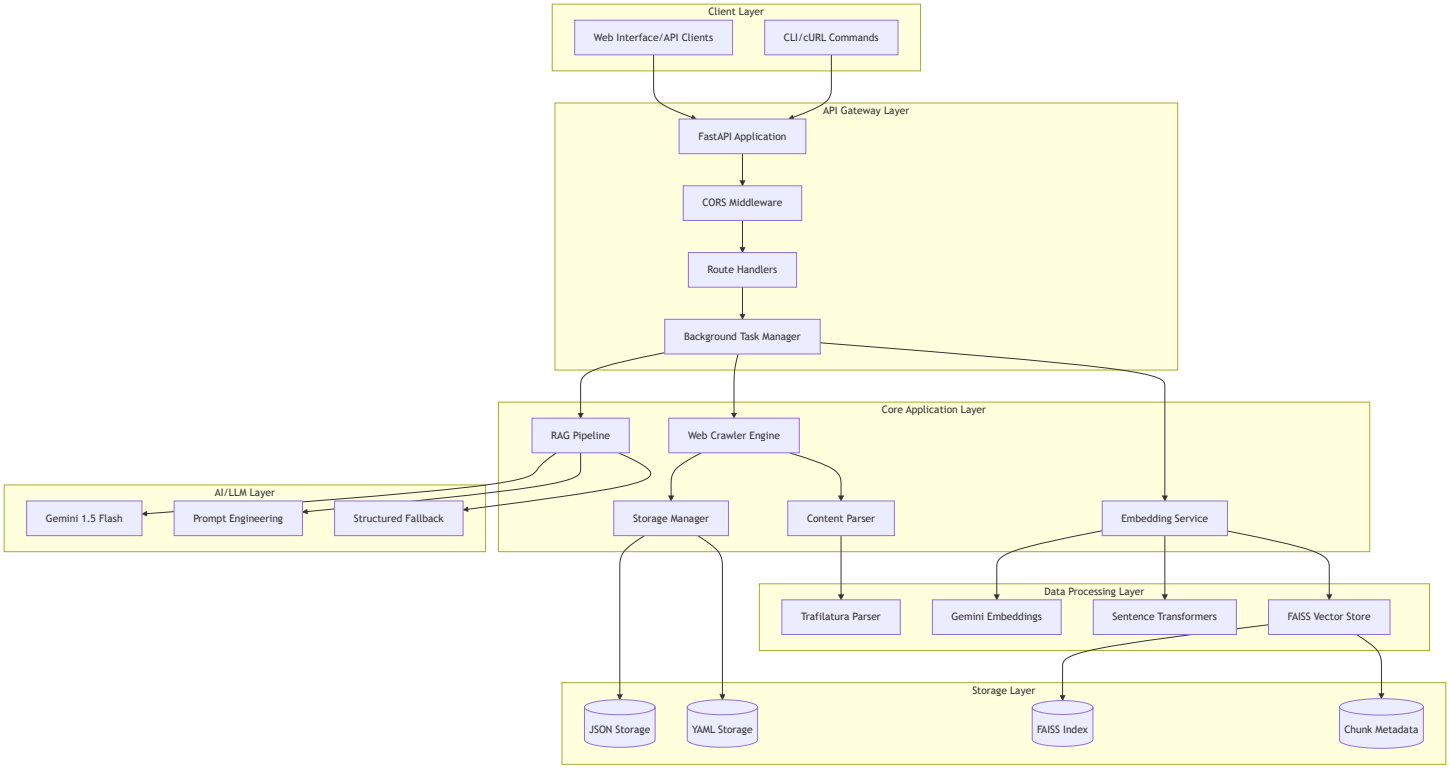
AI-Powered Documentation Crawler - Micro-Level Architecture System

Executive Summary

The AI-Powered Documentation Crawler is a comprehensive RAG (Retrieval-Augmented Generation) system built on FastAPI that crawls documentation websites, processes content through advanced embeddings, and provides intelligent Q/A capabilities using Google's Gemini LLM with sophisticated fallback mechanisms.

1. System Architecture Overview

1.1 High-Level Architecture

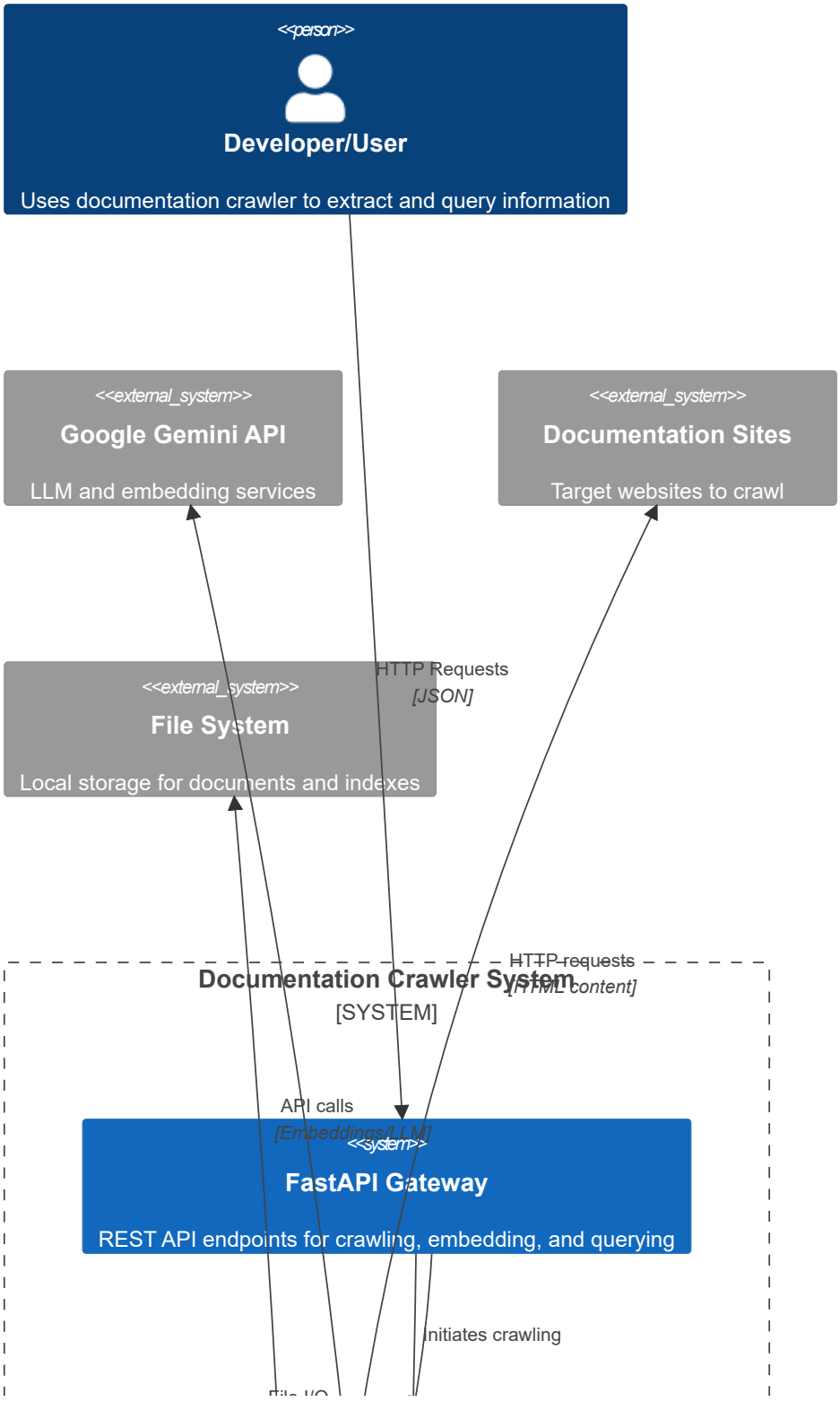


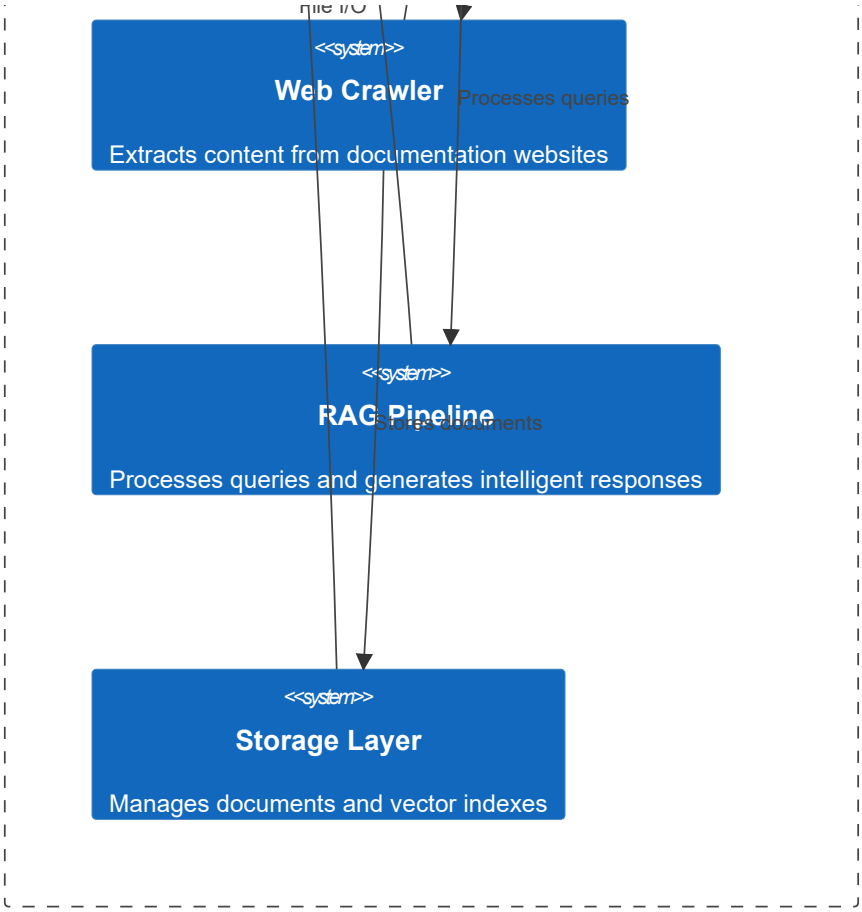
1.2 Technology Stack

Layer	Technology	Purpose
API Framework	FastAPI	Async REST API with automatic documentation
Web Crawling	aiohttp, trafilatura	Async HTTP requests and content extraction
Storage	JSON, YAML, FAISS	Dual-format storage with vector indexing
Embeddings	Google Generative AI, sentence-transformers	Primary + fallback embedding generation
Vector Search	FAISS	High-performance similarity search
LLM	Google Gemini 1.5 Flash	Advanced language model for responses
Configuration	python-dotenv, Pydantic	Environment and settings management
Logging	Python logging	Structured application logging

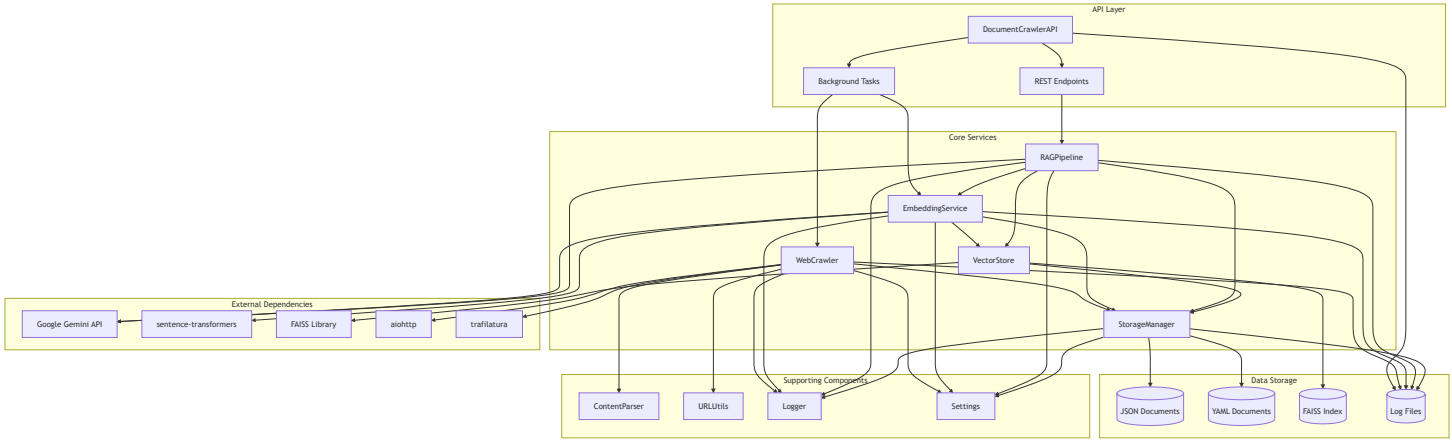
1.3 High-Level System Overview

AI-Powered Documentation Crawler - System Context

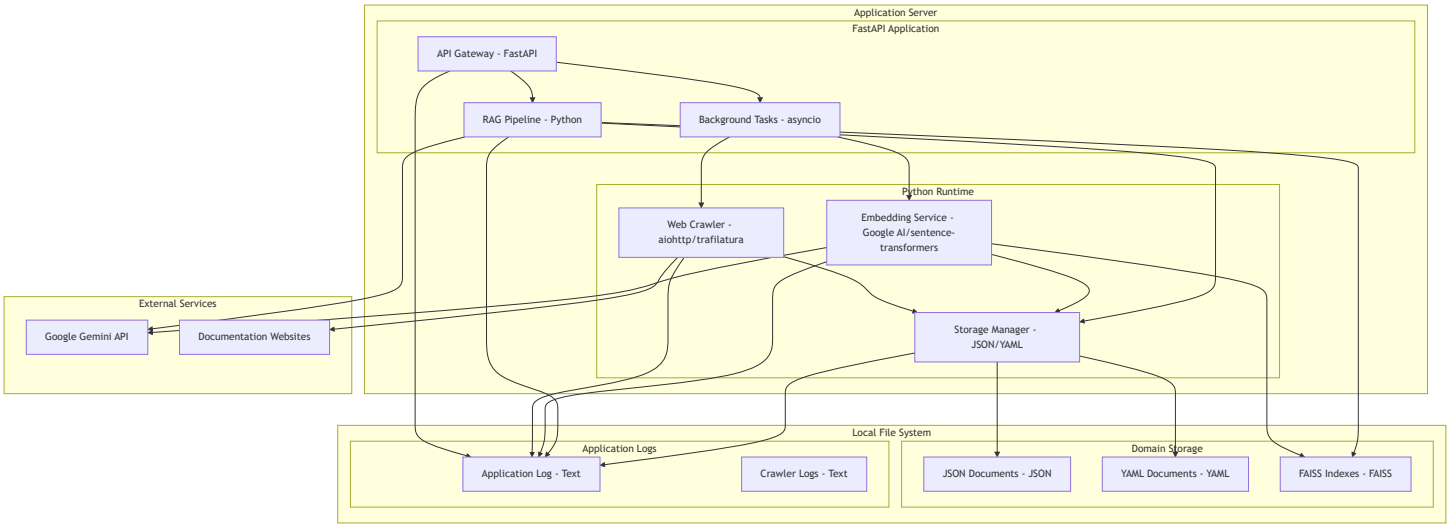




1.4 Service Architecture Overview

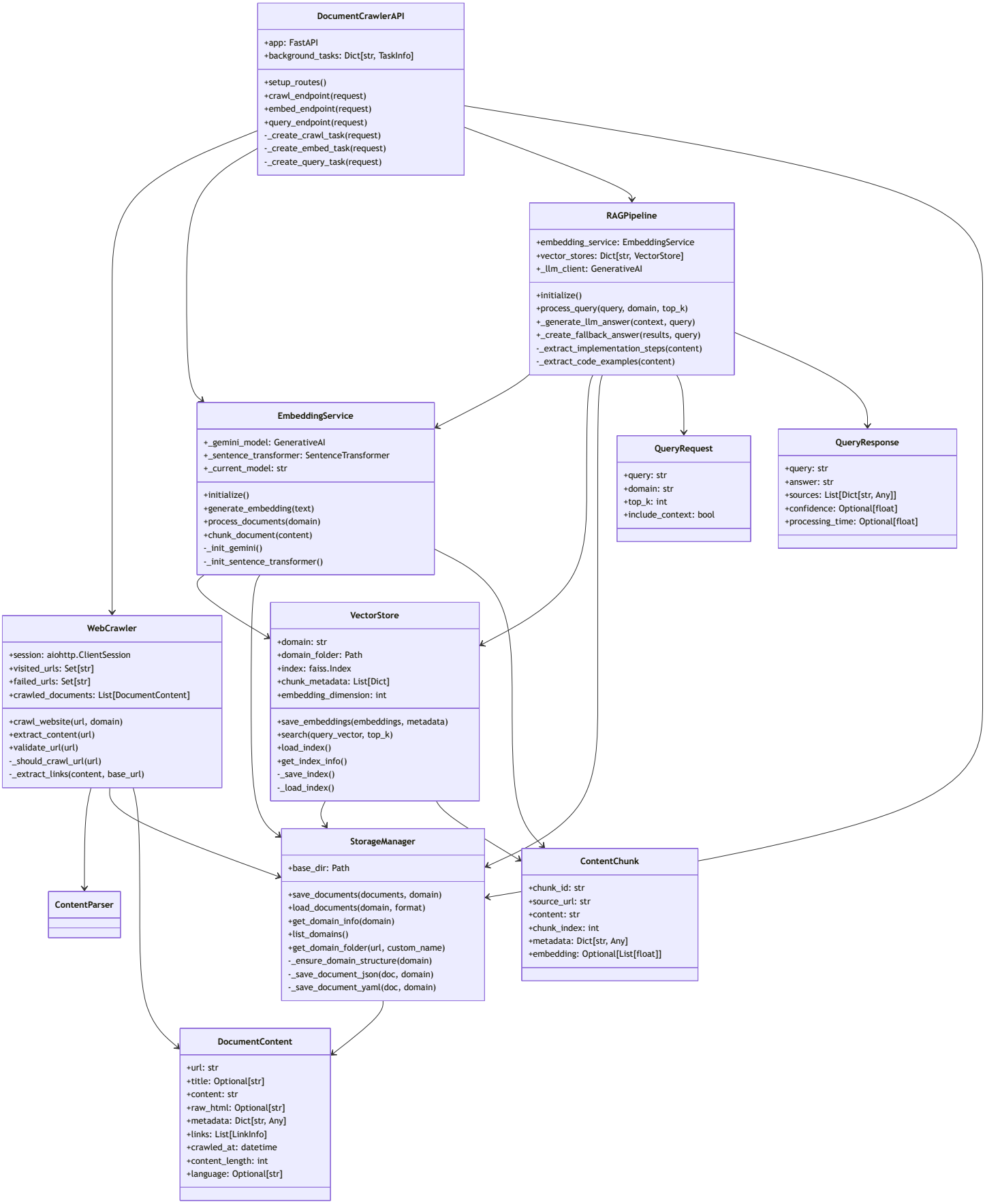


1.5 Deployment Architecture



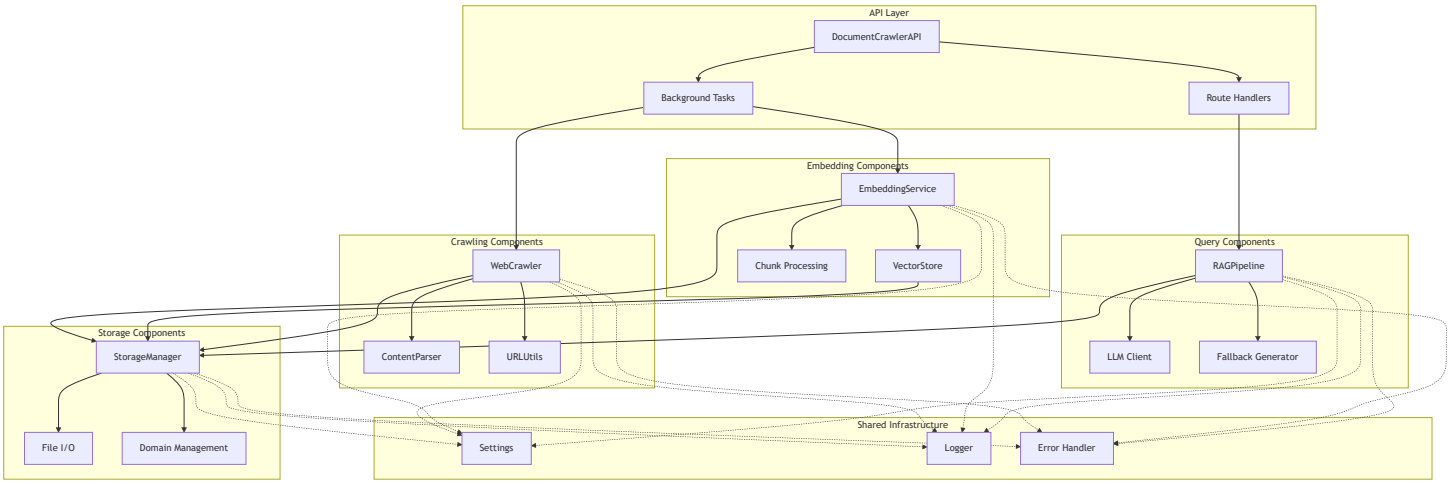
2. Abstract Services Layer

2.1 Core Classes & Data Models


DocumentCrawlerAPI
+app: FastAPI
+background_tasks: Dict[str, TaskInfo]
+setup_routes()
+crawl_endpoint(request)
+embed_endpoint(request)
+query_endpoint(request)
-_create_crawl_task(request)
-_create_embed_task(request)
-_create_query_task(request)
RAGPipeline
+embedding_service: EmbeddingService
+vector_stores: Dict[str, VectorStore]
+_llm_client: GenerativeAI
+initialize()
+process_query(query, domain, top_k)
+_generate_llm_answer(context, query)
+_create_fallback_answer(results, query)
-_extract_implementation_steps(content)
-_extract_code_examples(content)
EmbeddingService
+_gemini_model: GenerativeAI
+_sentence_transformer: SentenceTransformer
+_current_model: str
+initialize()
+generate_embedding(text)
+process_documents(domain)
+chunk_document(content)
-_init_gemini()
-_init_sentence_transformer()
QueryRequest
+query: str
+domain: str
+top_k: int
+include_context: bool
QueryResponse
+query: str
+answer: str
+sources: List[Dict[str, Any]]
+confidence: Optional[float]
+processing_time: Optional[float]
WebCrawler
+session: aiohttp.ClientSession
+visited_urls: Set[str]
+failed_urls: Set[str]
+crawled_documents: List[DocumentContent]
+crawl_website(url, domain)
+extract_content(url)
+validate_url(url)
-_should_crawl_url(url)
-_extract_links(content, base_url)
VectorStore
+domain: str
+domain_folder: Path
+index: faiss.Index
+chunk_metadata: List[Dict]
+embedding_dimension: int
+save_embeddings(embeddings, metadata)
+search(query_vector, top_k)
+load_index()
+get_index_info()
-_save_index()
-_load_index()
StorageManager
+base_dir: Path
+save_documents(documents, domain)
+load_documents(domain, format)
+get_domain_info(domain)
+list_domains()
+get_domain_folder(url, custom_name)
-_ensure_domain_structure(domain)
-_save_document_json(doc, domain)
-_save_document_yaml(doc, domain)
ContentChunk
+chunk_id: str
+source_url: str
+content: str
+chunk_index: int
+metadata: Dict[str, Any]
+embedding: Optional[List[float]]
ContentParser
DocumentContent
+url: str
+title: Optional[str]
+content: str
+raw_html: Optional[str]
+metadata: Dict[str, Any]
+links: List[LinkInfo]
+crawled_at: datetime
+content_length: int
+language: Optional[str]

2.2 Service Boundaries & Responsibilities

2.2.1 Core Component Boundaries



2.2.2 Component Dependencies

```
# Component Dependency Graph
COMPONENT_DEPENDENCIES = {
    "DocumentCrawlerAPI": {
        "depends_on": ["WebCrawler", "EmbeddingService", "RAGPipeline", "StorageManager"],
        "provides": ["REST API endpoints", "Background task management"],
        "scope": "Application"
    },

    "WebCrawler": {
        "depends_on": ["ContentParser", "StorageManager", "URLUtils", "aiohttp", "trafilatura"],
        "provides": ["Website crawling", "Content extraction", "URL management"],
        "scope": "Domain"
    },

    "EmbeddingService": {
        "depends_on": ["VectorStore", "StorageManager", "Google Generative AI", "sentence-trans
        "provides": ["Text embedding", "Document chunking", "Vector generation"],
        "scope": "Domain"
    },

    "RAGPipeline": {
        "depends_on": ["VectorStore", "EmbeddingService", "Google Generative AI"],
        "provides": ["Query processing", "Answer generation", "Context retrieval"],
        "scope": "Domain"
    },

    "VectorStore": {
        "depends_on": ["FAISS", "StorageManager"],
        "provides": ["Vector similarity search", "Index management", "Metadata handling"],
        "scope": "Infrastructure"
    },

    "StorageManager": {
        "depends_on": ["File System", "JSON/YAML parsers"],
        "provides": ["Document persistence", "Domain organization", "Format management"],
        "scope": "Infrastructure"
    }
}
```

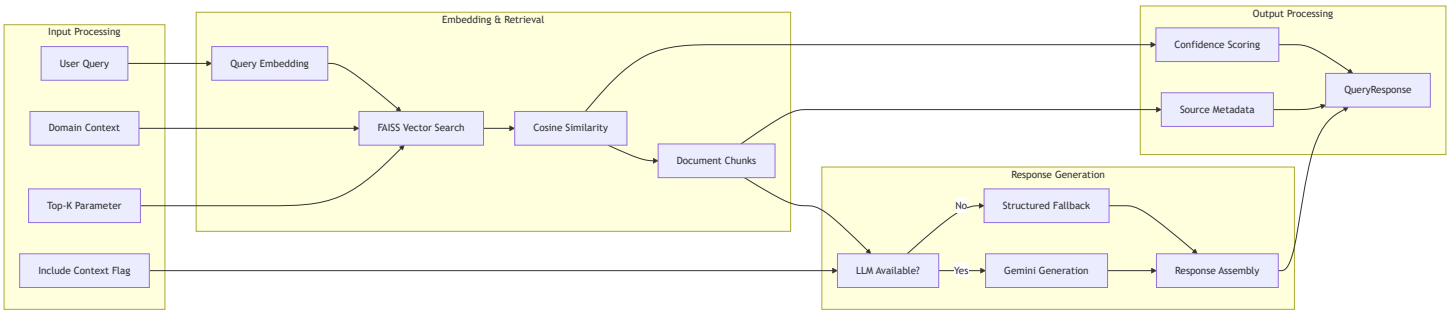
2.3 Data Contracts & Models

```python
# Actual Pydantic Data Models from schemas.py
from typing import List, Dict, Optional, Any
from pydantic import BaseModel, HttpUrl, Field
from datetime import datetime

class LinkInfo(BaseModel):
    """Information about a link found in a document."""
    url: str
    text: Optional[str] = None
    is_internal: bool = Field(default=False, description="Whether link is internal to docs doma
    is_visited: bool = Field(default=False, description="Whether link has been crawled")

class DocumentContent(BaseModel):
    """Structured content of a crawled document."""
    url: str
    title: Optional[str] = None
    content: str = Field(description="Main text content extracted from the page")
    raw_html: Optional[str] = None
    metadata: Dict[str, Any] = Field(default_factory=dict)
    links: List[LinkInfo] = Field(default_factory=list)
    crawled_at: datetime = Field(default_factory=datetime.now)
    content_length: int = Field(default=0, description="Length of extracted content")
    language: Optional[str] = None

class CrawlSession(BaseModel):
    """Information about a crawling session."""
    domain: str
    start_url: str
    domain_folder: str = Field(description="Folder name for storing domain data")
    started_at: datetime = Field(default_factory=datetime.now)
    completed_at: Optional[datetime] = None
    total_pages: int = Field(default=0)
    successful_pages: int = Field(default=0)
    failed_pages: int = Field(default=0)
    status: str = Field(default="in_progress")  # in_progress, completed, failed

class ContentChunk(BaseModel):
    """A chunk of content for embedding."""
    chunk_id: str = Field(description="Unique identifier for the chunk")
    source_url: str = Field(description="URL of the source document")
    content: str = Field(description="Text content of the chunk")
```

```python
    chunk_index: int = Field(description="Position of chunk in the document")
    metadata: Dict[str, Any] = Field(default_factory=dict)
    embedding: Optional[List[float]] = None

class EmbeddingIndex(BaseModel):
    """Information about an embedding index."""
    domain: str
    total_chunks: int = Field(default=0)
    vector_dimension: int = Field(default=384)
    model_name: str = Field(description="Name of the embedding model used")
    created_at: datetime = Field(default_factory=datetime.now)
    index_file_path: str = Field(description="Path to FAISS index file")
    metadata_file_path: str = Field(description="Path to chunk metadata file")

class QueryRequest(BaseModel):
    """Request model for Q/A queries."""
    query: str = Field(description="Natural language query")
    domain: str = Field(description="Domain to search in")
    top_k: int = Field(default=5, description="Number of relevant chunks to retrieve")
    include_context: bool = Field(default=True, description="Whether to include source context")

class QueryResponse(BaseModel):
    """Response model for Q/A queries."""
    query: str
    answer: str
    sources: List[Dict[str, Any]] = Field(default_factory=list)
    confidence: Optional[float] = None
    processing_time: Optional[float] = None
```

3. RAG Pipeline Architecture (Core Focus)

2.1 RAG Component Breakdown



2.2 RAG Pipeline Components Detail

2.2.1 Query Processing Flow

```
# Query Processing Pipeline
QUERY_PIPELINE = {
    "input_validation": "Domain existence check",
    "embedding_generation": "EmbeddingService.generate_embeddings()",
    "vector_search": "VectorStore.search(query_embedding, top_k)",
    "context_preparation": "Format chunks with metadata",
    "response_generation": "LLM or Fallback based on availability"
}
```

2.2.2 Vector Search Process

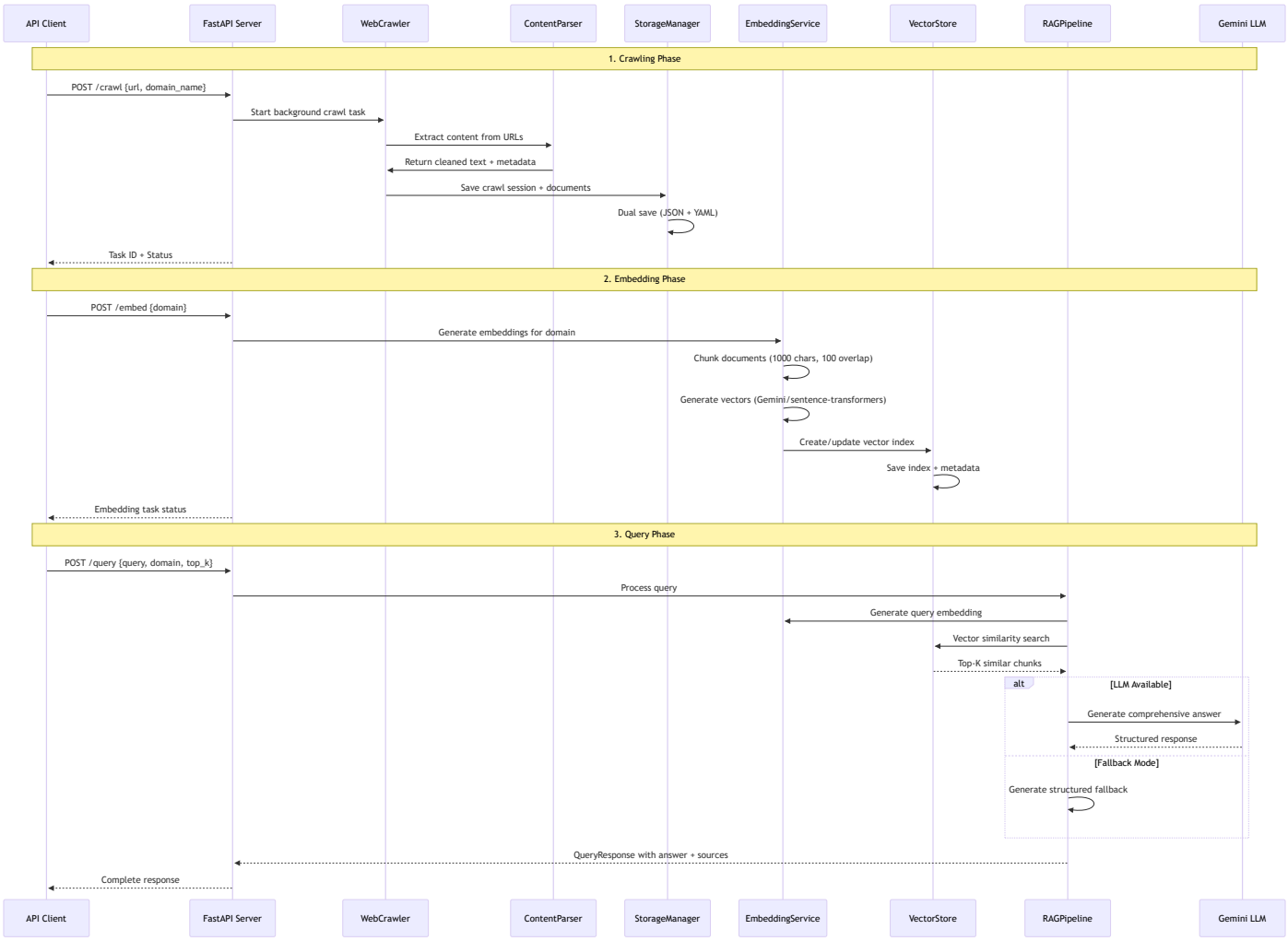
```
# FAISS Vector Search Flow
SEARCH_PROCESS = {
    "query_embedding": "768-dimensional vector (Gemini) or 384-d (sentence-transformers)",
    "normalization": "L2 normalization for cosine similarity",
    "similarity_calculation": "Inner product (IndexFlatIP)",
    "top_k_retrieval": "Configurable (default: 5)",
    "metadata_enrichment": "Source URLs, titles, chunk indices"
}
```

2.2.3 Dual Response Generation

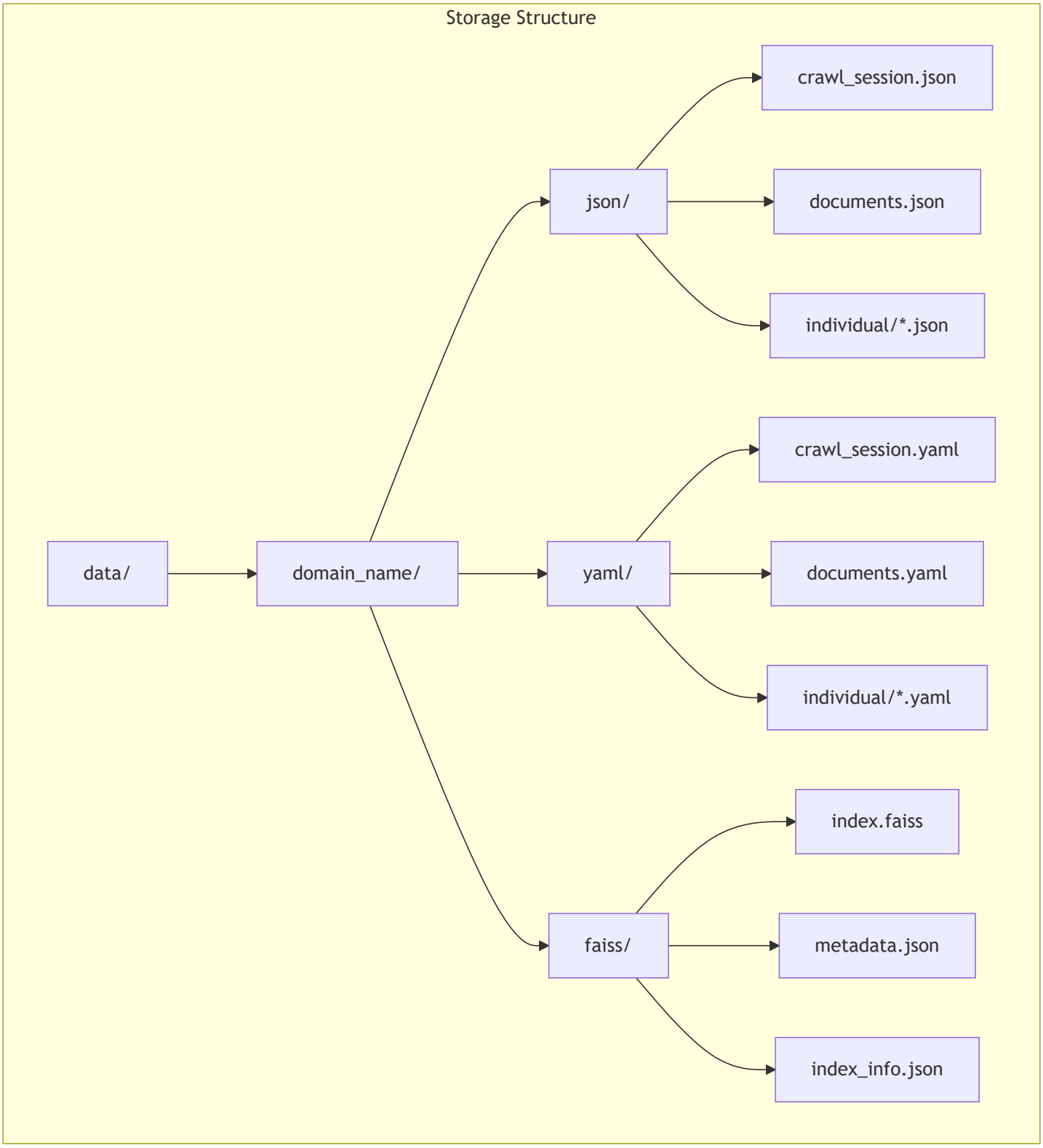
```
# Response Generation Strategy
GENERATION_MODES = {
    "llm_mode": {
        "condition": "GEMINI_API_KEY available AND include_context=True",
        "engine": "Gemini 1.5 Flash",
        "prompt": "Senior engineer-level comprehensive prompt",
        "max_tokens": "4096",
        "temperature": "0.3"
    },
    "fallback_mode": {
        "condition": "LLM unavailable OR include_context=False",
        "engine": "Structured content processing",
        "features": ["TL;DR", "Overview", "Key Information", "Implementation Steps", "Sources"],
        "format": "Markdown with sections and confidence indicators"
    }
}
```

3. Data Flow Architecture

3.1 Complete System Data Flow



3.2 Domain-Based Storage Architecture



4. Component Deep Dive

4.1 Web Crawler Engine (`WebCrawler`)

```
# Crawler Component Architecture
CRAWLER_COMPONENTS = {
    "session_management": "aiohttp.ClientSession with connection pooling",
    "url_handling": "URLUtils for normalization and validation",
    "content_extraction": "ContentParser with trafilatura",
    "concurrency_control": "Semaphore-based request limiting",
    "domain_restriction": "Base domain validation and filtering",
    "state_tracking": {
        "visited_urls": "Set for cycle prevention",
        "failed_urls": "Set for error tracking",
        "crawled_documents": "List of processed content"
    }
}

# Crawler Configuration
CRAWLER_CONFIG = {
    "max_concurrent_requests": 10,
    "request_timeout": 30,
    "retry_attempts": 3,
    "delay_between_requests": 1.0,
    "depth_limit": "configurable (default: unlimited)"
}
```

4.2 Storage Manager (`StorageManager`)

```
# Storage Manager Architecture
STORAGE_FEATURES = {
    "dual_format": "Simultaneous JSON and YAML storage",
    "domain_organization": "Separate folders per crawled domain",
    "individual_files": "Per-document storage in individual/ subdirs",
    "metadata_preservation": "Complete URL, title, and content metadata",
    "session_tracking": "Crawl session information and statistics"
}

# Storage Operations
STORAGE_OPERATIONS = {
    "save_crawl_session": "Store crawling metadata and statistics",
    "save_documents": "Bulk document storage with dual format",
    "load_documents": "Retrieve documents by domain and format",
    "list_domains": "Get all crawled domains with stats",
    "get_domain_folder": "Create/verify domain folder structure"
}
```

4.3 Embedding Service (`EmbeddingService`)

```
# Embedding Service Architecture
EMBEDDING_STRATEGY = {
    "primary_provider": {
        "service": "Google Generative AI",
        "model": "models/embedding-001",
        "dimension": 768,
        "api_key_required": True
    },
    "fallback_provider": {
        "service": "sentence-transformers",
        "model": "all-MiniLM-L6-v2",
        "dimension": 384,
        "local_execution": True
    },
    "chunking_strategy": {
        "chunk_size": 1000,
        "chunk_overlap": 100,
        "max_chunks_per_doc": 50
    }
}
```

4.4 Vector Store (`VectorStore`)

```
# FAISS Vector Store Implementation
VECTOR_STORE_CONFIG = {
    "index_type": "IndexFlatIP (Inner Product for cosine similarity)",
    "storage_format": "Binary FAISS index + JSON metadata",
    "normalization": "L2 normalization for consistent similarity scores",
    "metadata_tracking": {
        "source_url": "Original document URL",
        "title": "Document title",
        "content": "Chunk content",
        "chunk_index": "Position within document"
    }
}

# Search Operations
SEARCH_CAPABILITIES = {
    "similarity_metric": "Cosine similarity via inner product",
    "top_k_retrieval": "Configurable result count (1-20)",
    "score_normalization": "Similarity scores between 0-1",
    "metadata_enrichment": "Full context with each result"
}
```

5. Advanced RAG Features

5.1 Comprehensive Prompt Engineering

```
# Senior Engineer Prompt Template
PROMPT_STRUCTURE = {
    "role_definition": "Expert technical assistant (senior engineer/architect)",
    "output_format": "Complete, actionable, copy-pasteable technical guidance",
    "required_sections": [
        "TL;DR (1-3 sentence summary)",
        "Overview (architectural summary)",
        "Prerequisites (exact requirements)",
        "Architecture (diagrams + data flow)",
        "Step-by-step Implementation (numbered with verification)",
        "Example (minimal, runnable)",
        "Configuration & Secrets (.env keys, tokens)",
        "Advanced/Production Notes (scaling, monitoring)",
        "Troubleshooting (common failures + fixes)",
        "Next steps (3-6 concrete follow-ups)",
        "Checklist (actionable items)"
    ],
    "deliverables": [
        "Copy & Paste runnable section",
        "Verification block (3 quick checks)"
    ]
}
```

5.2 Intelligent Fallback System

```
# Structured Fallback Generation
FALLBACK_FEATURES = {
    "content_analysis": {
        "main_topic_extraction": "Heuristic-based topic identification",
        "implementation_detection": "Keywords for setup/config content",
        "code_example_extraction": "Regex patterns for code blocks",
        "step_identification": "Numbered/bulleted step recognition"
    },
    "response_structure": {
        "tl_dr": "Auto-generated summary with source count",
        "overview": "Combined content from top results",
        "key_information": "Structured presentation of top 3 sources",
        "implementation_steps": "Extracted or generic actionable steps",
        "code_examples": "Formatted code blocks with language detection",
        "next_steps": "Context-aware recommendations",
        "sources": "Complete source list with relevance scores"
    },
    "confidence_indicators": {
        "high": "Score > 0.8 - Strong match found",
        "moderate": "Score > 0.6 - Good match, consider refinement",
        "low": "Score ≤ 0.6 - Limited matches, rephrase query"
    }
}
```

6. API Endpoint Architecture

6.1 Complete API Surface

```
# FastAPI Endpoint Mapping
API_ENDPOINTS = {
    "health": {
        "method": "GET",
        "path": "/",
        "function": "health_check",
        "returns": "System status + available domains"
    },
    "crawl": {
        "method": "POST",
        "path": "/crawl",
        "function": "crawl_documentation",
        "background_task": "_crawl_task",
        "returns": "Task ID + crawl session info"
    },
    "embed": {
        "method": "POST",
        "path": "/embed",
        "function": "generate_embeddings",
        "background_task": "_embed_task",
        "returns": "Task ID + embedding status"
    },
    "query": {
        "method": "POST",
        "path": "/query",
        "function": "query_documentation",
        "returns": "Answer + sources + confidence + timing"
    },
    "domains": {
        "method": "GET",
        "path": "/domains",
        "function": "list_domains",
        "returns": "Domain info + embedding status"
    },
    "documents": {
        "method": "GET",
        "path": "/domains/{domain_name}/documents",
        "function": "get_domain_documents",
```

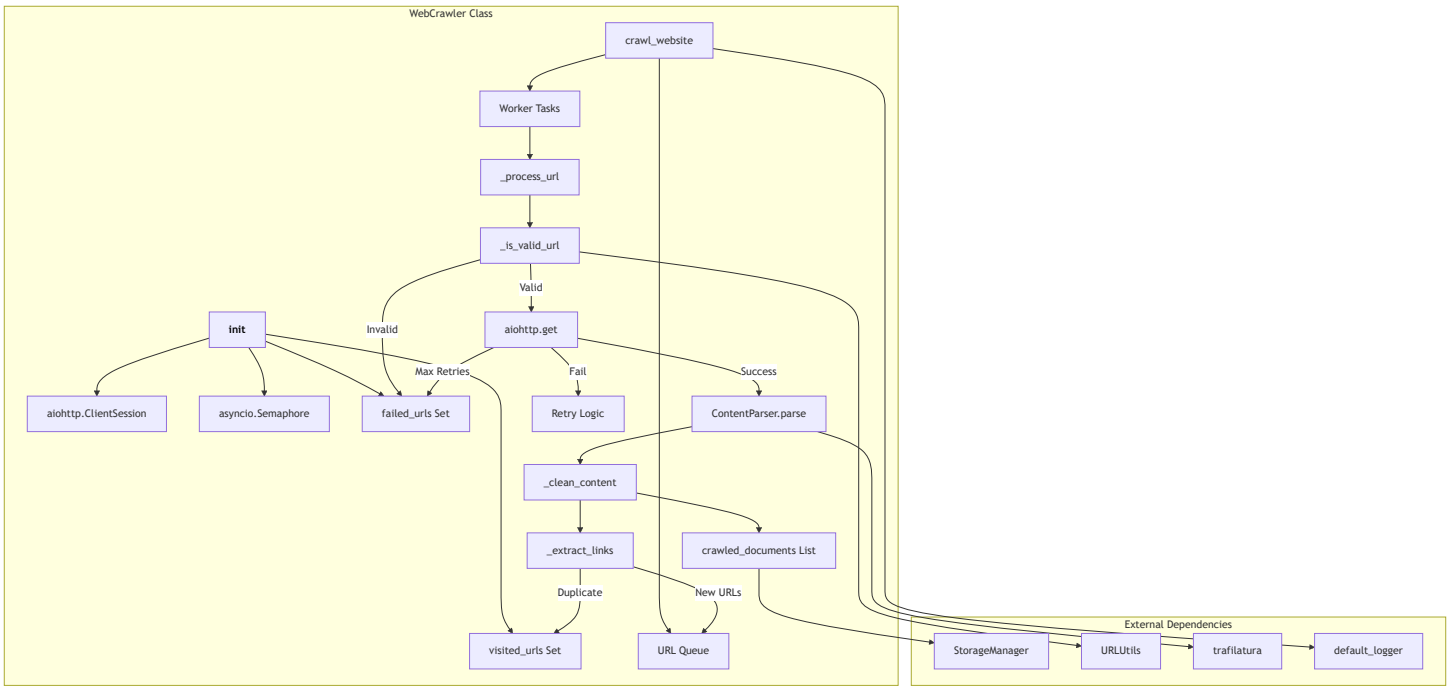
```
        "returns": "Document list by format (JSON/YAML)"
    },
    "task_status": {
        "method": "GET",
        "path": "/tasks/{task_id}",
        "function": "get_task_status",
        "returns": "Background task progress"
    },
    "system_status": {
        "method": "GET",
        "path": "/status",
        "function": "get_system_status",
        "returns": "Pipeline info + domain stats"
    }
}
```

6.2 Background Task Management

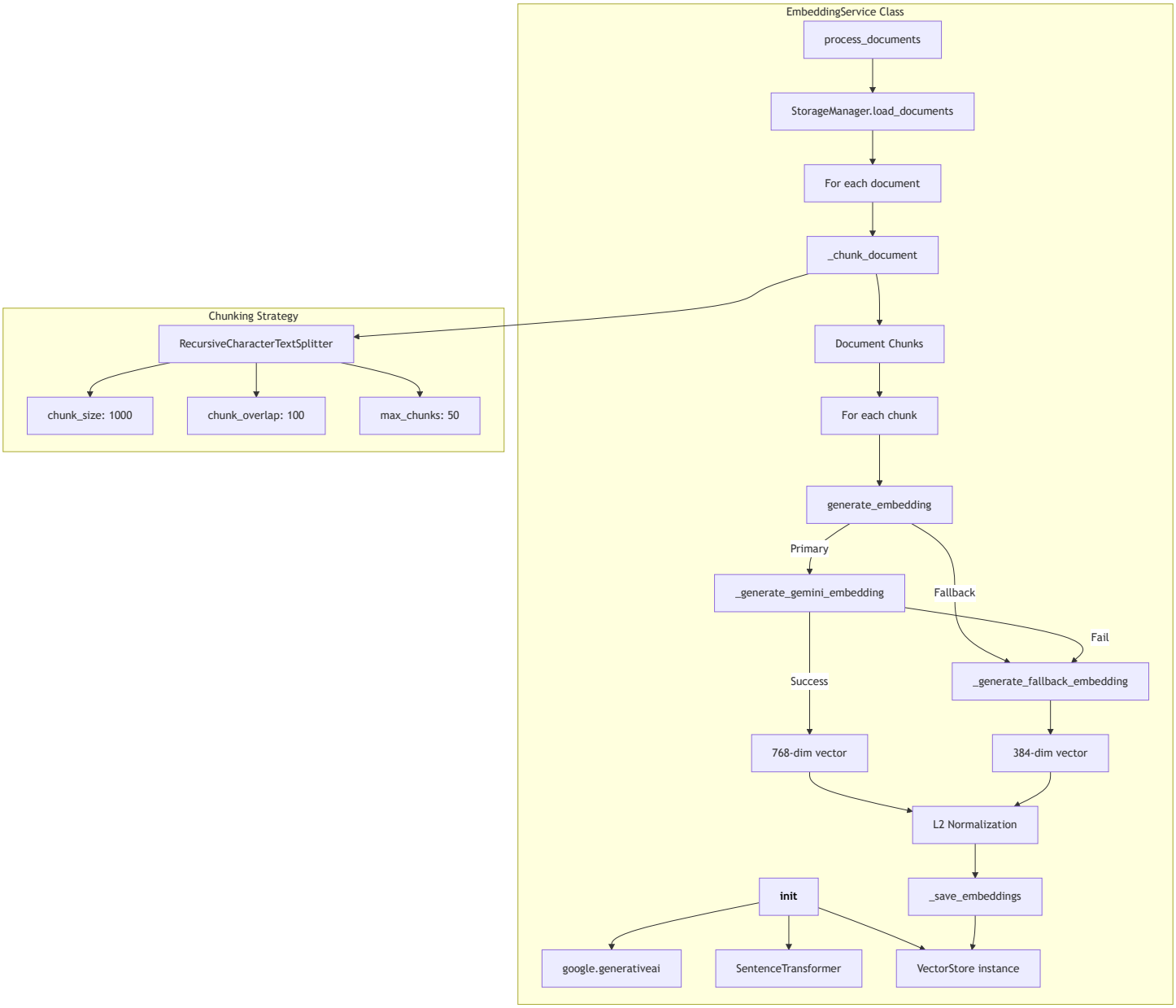
```
# Background Task Architecture
TASK_MANAGEMENT = {
    "task_tracking": {
        "storage": "In-memory dictionary with UUID keys",
        "metadata": ["type", "status", "start_time", "domain", "progress"],
        "states": ["started", "processing", "completed", "failed"]
    },
    "crawl_tasks": {
        "function": "_crawl_task",
        "operations": ["URL validation", "WebCrawler execution", "Document storage"],
        "error_handling": "Comprehensive exception capture + logging"
    },
    "embed_tasks": {
        "function": "_embed_task",
        "operations": ["Document loading", "Chunking", "Embedding generation", "FAISS indexing"]
        "progress_tracking": "Chunk count + processing time"
    }
}
```

7. Micro-Level Component Diagrams

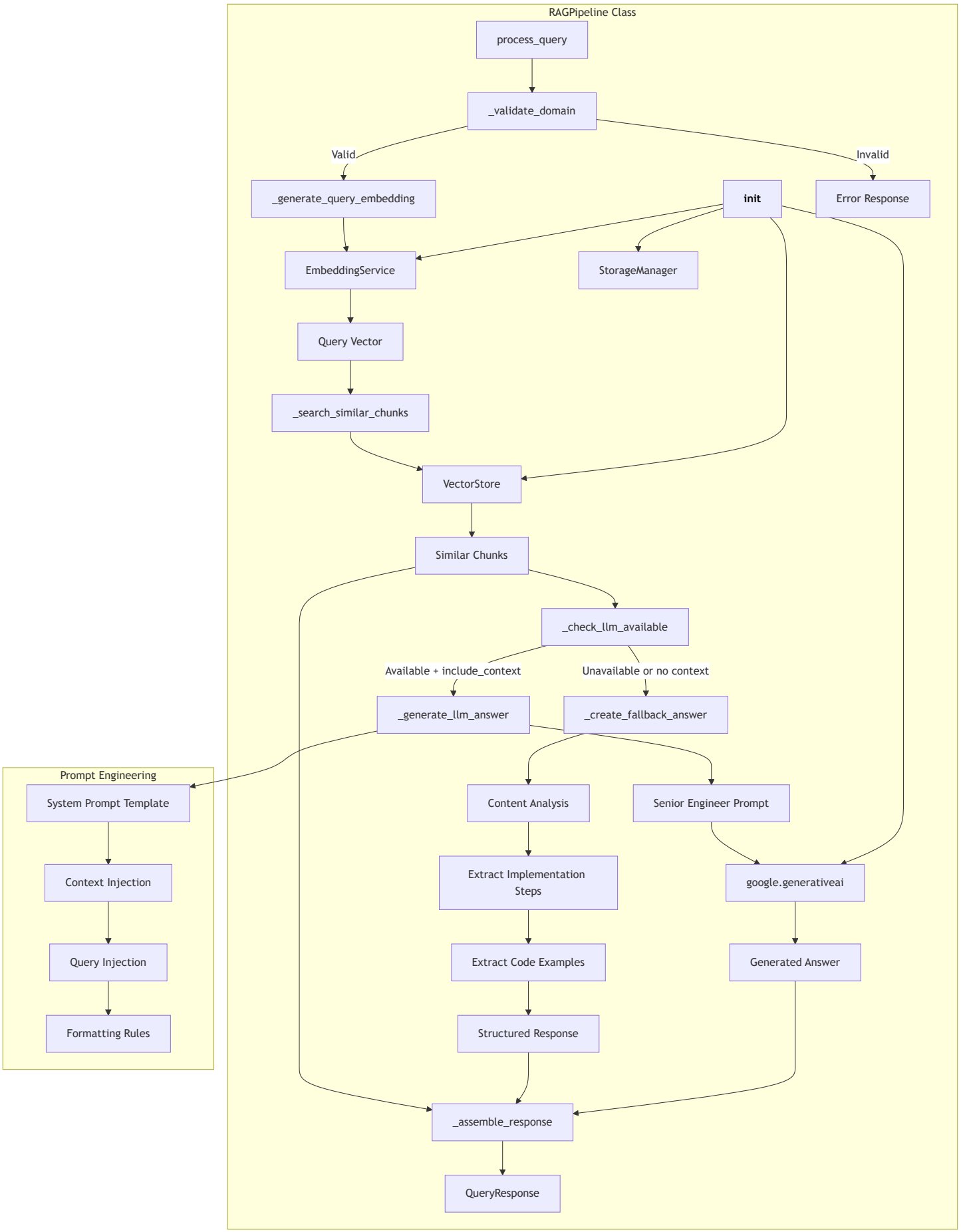
7.1 WebCrawler Internal Architecture



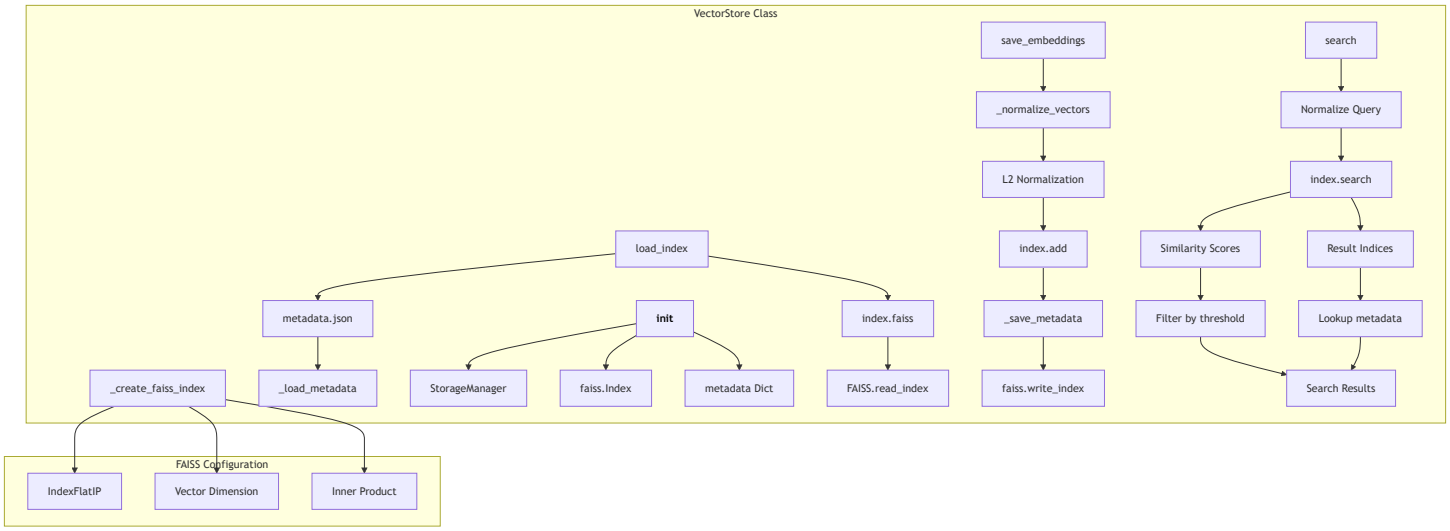
7.2 EmbeddingService Internal Architecture



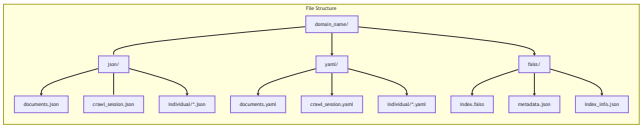
7.3 RAGPipeline Internal Architecture

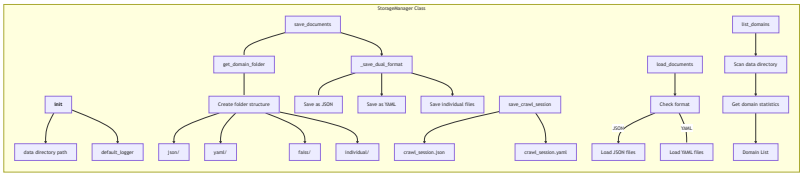


7.4 VectorStore Internal Architecture

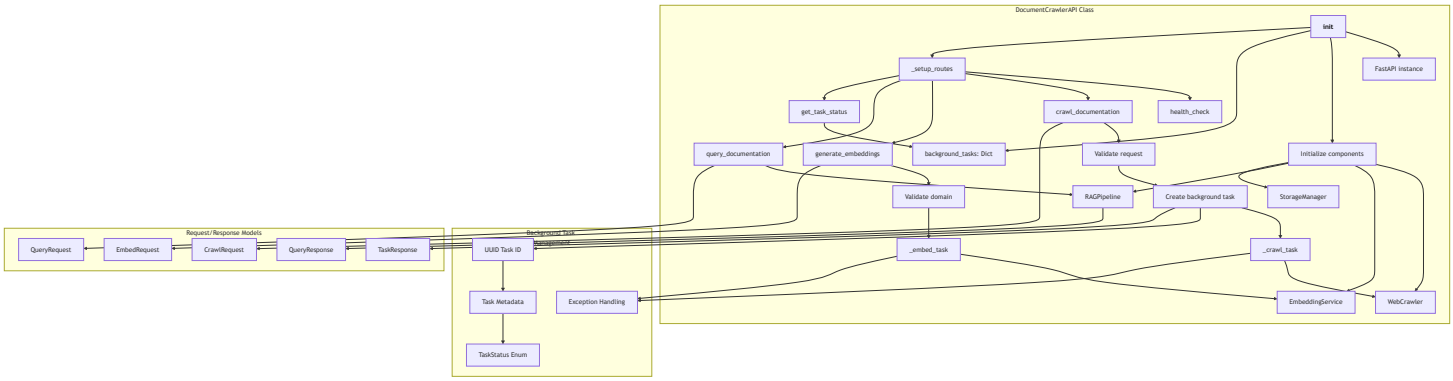


7.5 StorageManager Internal Architecture





7.6 DocumentCrawlerAPI Internal Architecture

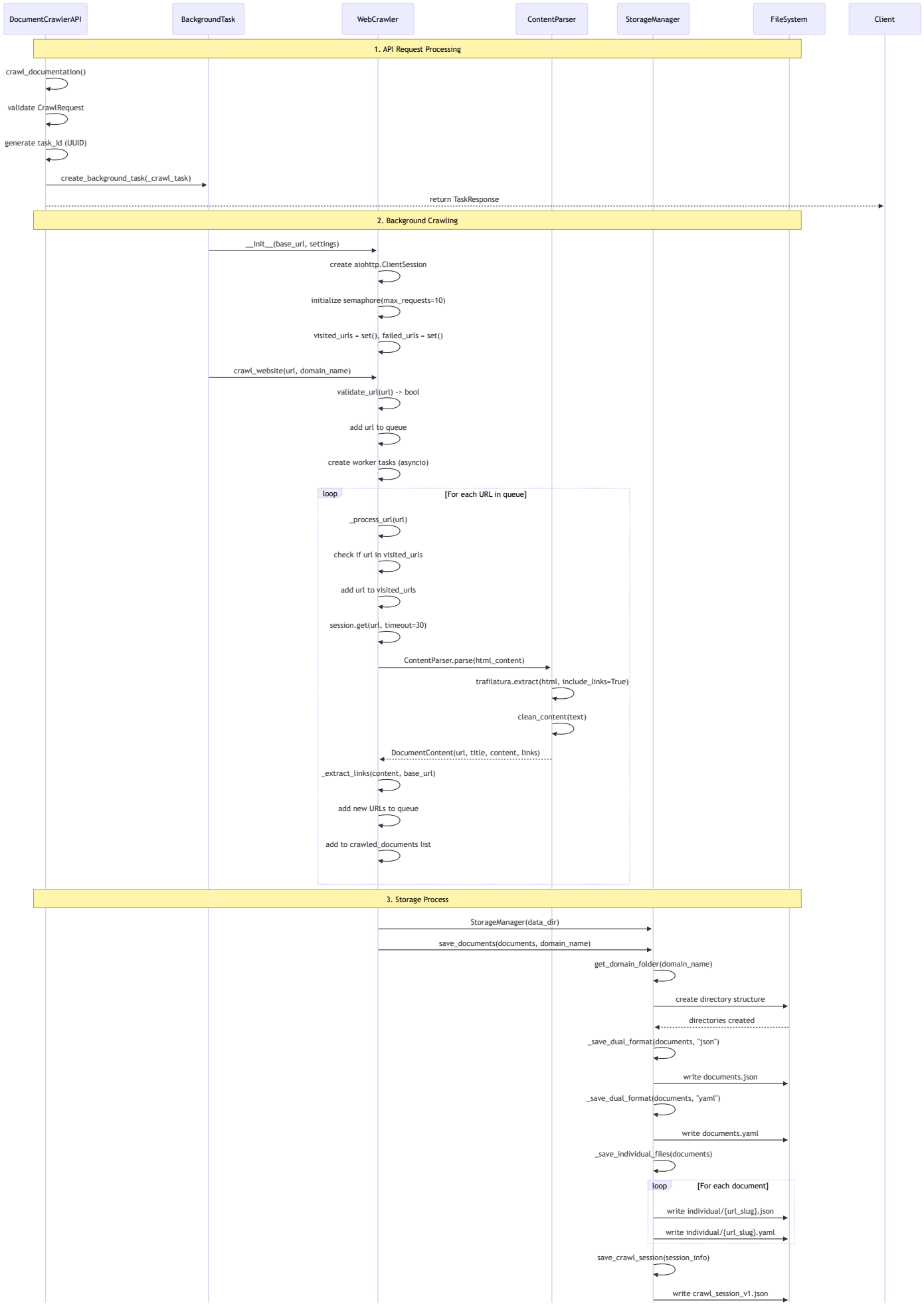


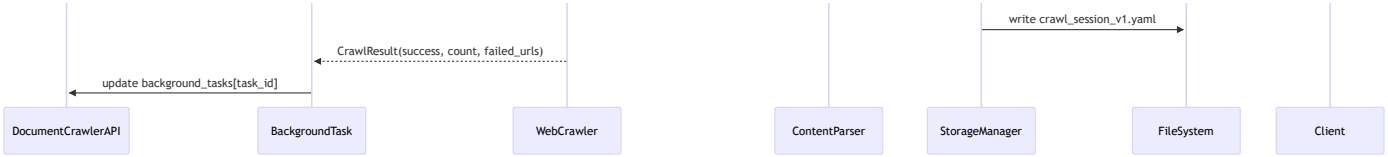
7.1 Settings Architecture

```
# Configuration System
SETTINGS_STRUCTURE = {
    "api_keys": {
        "GEMINI_API_KEY": "Google Generative AI authentication",
    },
    "crawler_settings": {
        "MAX_CONCURRENT_REQUESTS": 10,
        "REQUEST_TIMEOUT": 30,
        "RETRY_ATTEMPTS": 3,
        "DELAY_BETWEEN_REQUESTS": 1.0
    },
    "storage_settings": {
        "DATA_DIR": "./data",
        "LOG_LEVEL": "INFO"
    },
    "embedding_settings": {
        "CHUNK_SIZE": 1000,
        "CHUNK_OVERLAP": 100,
        "MAX_CHUNKS_PER_DOC": 50
    },
    "llm_settings": {
        "LLM_TEMPERATURE": 0.3,
        "LLM_MAX_TOKENS": 4096,
        "RAG_CONTEXT_CHUNKS": 5,
        "RAG_CONTENT_LENGTH": 1500
    },
    "faiss_settings": {
        "FAISS_INDEX_TYPE": "IndexFlatIP",
        "VECTOR_DIMENSION": "384 (fallback) / 768 (Gemini)"
    }
}
```

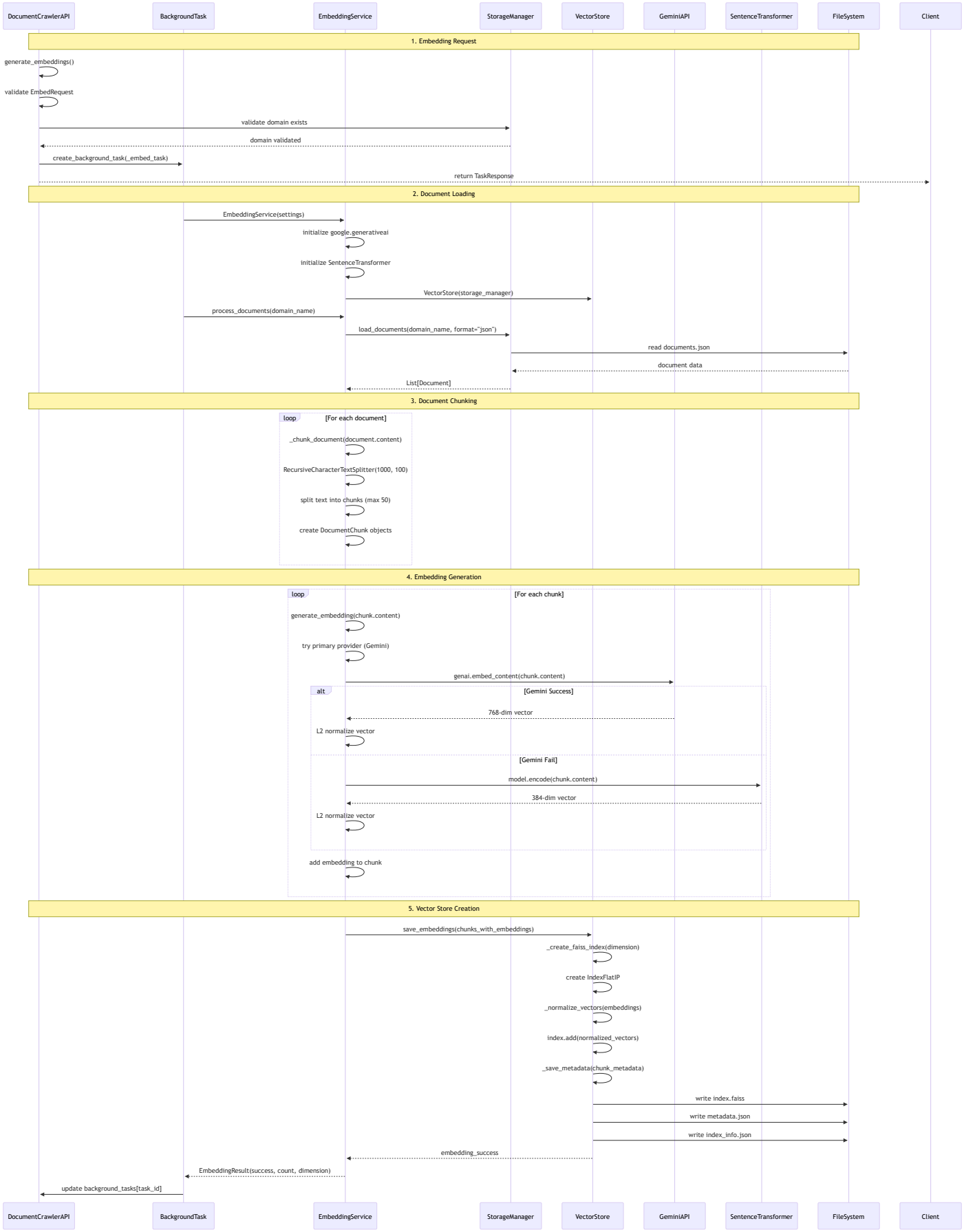
8. Detailed Function Call Flows

8.1 Complete Crawl Process Flow

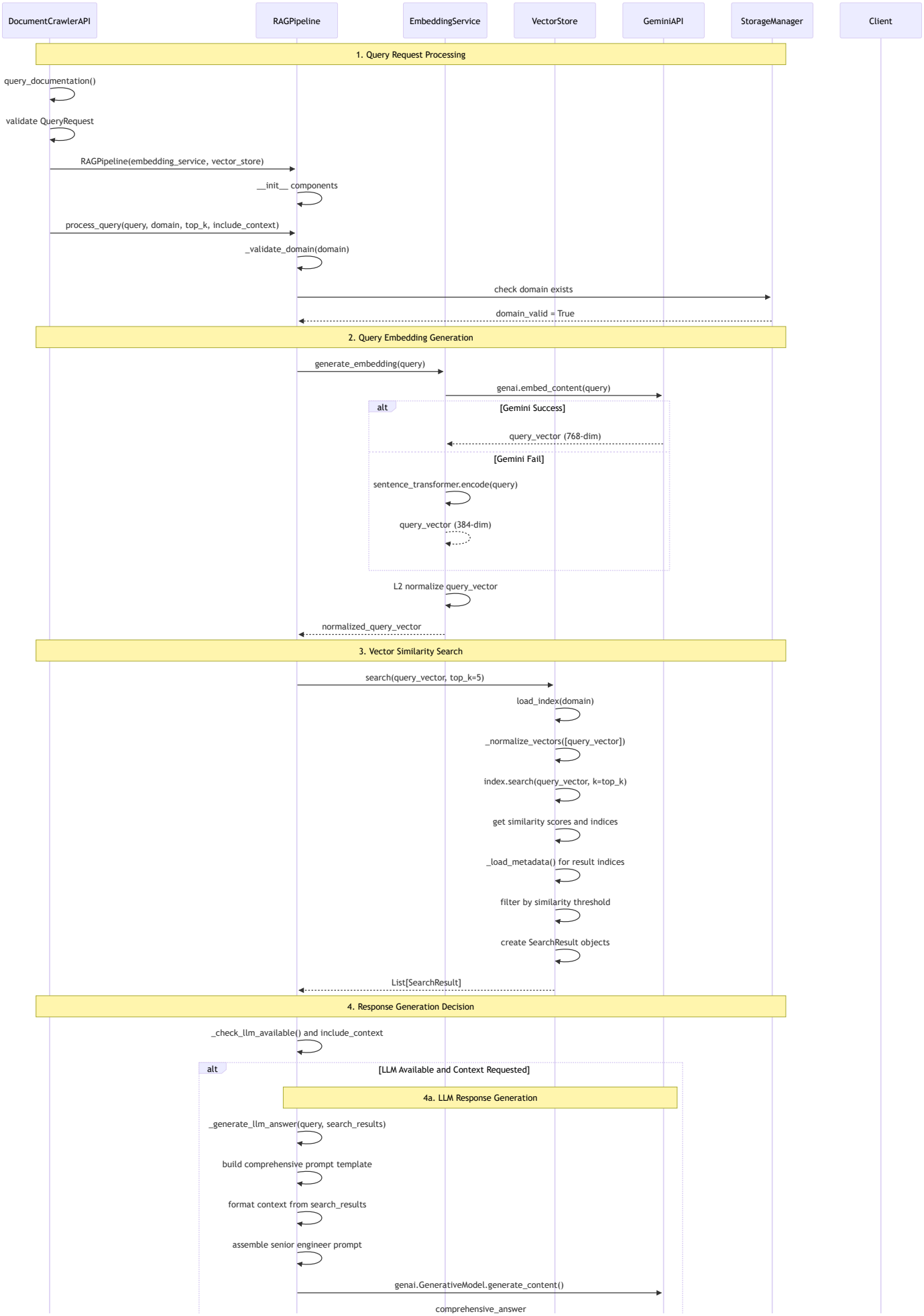

DocumentCrawlerAPI
BackgroundTask
WebCrawler
ContentParser
StorageManager
FileSystem
Client
1. API Request Processing
crawl_documentation()
validate CrawlRequest
generate task_id (UUID)
create_background_task(_crawl_task)
return TaskResponse
2. Background Crawling
__init__(base_url, settings)
create aiohttp.ClientSession
initialize semaphore(max_requests=10)
visited_urls = set(), failed_urls = set()
crawl_website(url, domain_name)
validate_url(url) -> bool
add url to queue
create worker tasks (asyncio)
loop
[For each URL in queue]
_process_url(url)
check if url in visited_urls
add url to visited_urls
session.get(url, timeout=30)
ContentParser.parse(html_content)
trafilatura.extract(html, include_links=True)
clean_content(text)
DocumentContent(url, title, content, links)
_extract_links(content, base_url)
add new URLs to queue
add to crawled_documents list
3. Storage Process
StorageManager(data_dir)
save_documents(documents, domain_name)
get_domain_folder(domain_name)
create directory structure
directories created
_save_dual_format(documents, "json")
write documents.json
_save_dual_format(documents, "yaml")
write documents.yaml
_save_individual_files(documents)
loop
[For each document]
write individual/{url_slug}.json
write individual/{url_slug}.yaml
save_crawl_session(session_info)
write crawl_session_v1.json

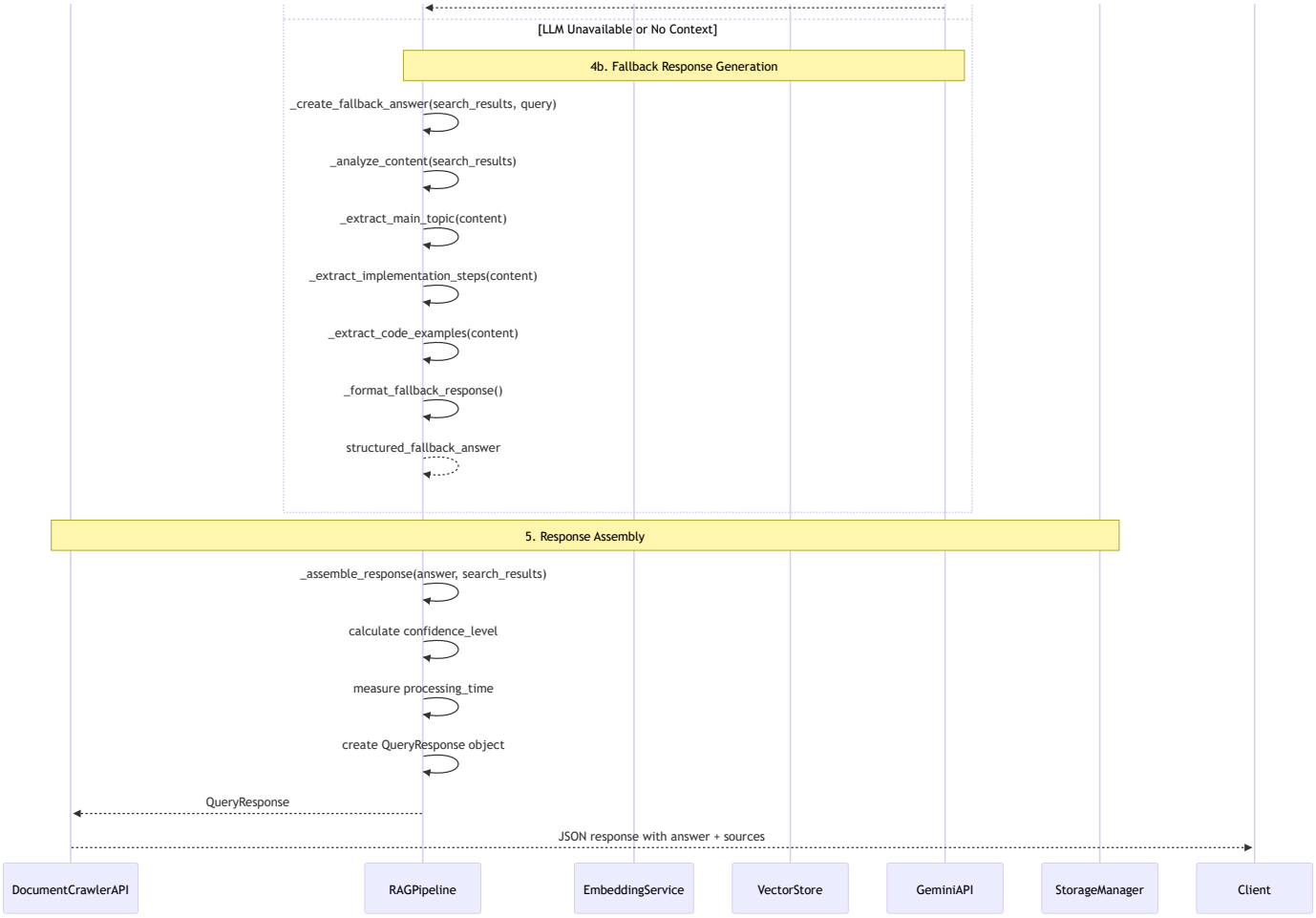


8.2 Complete Embedding Process Flow

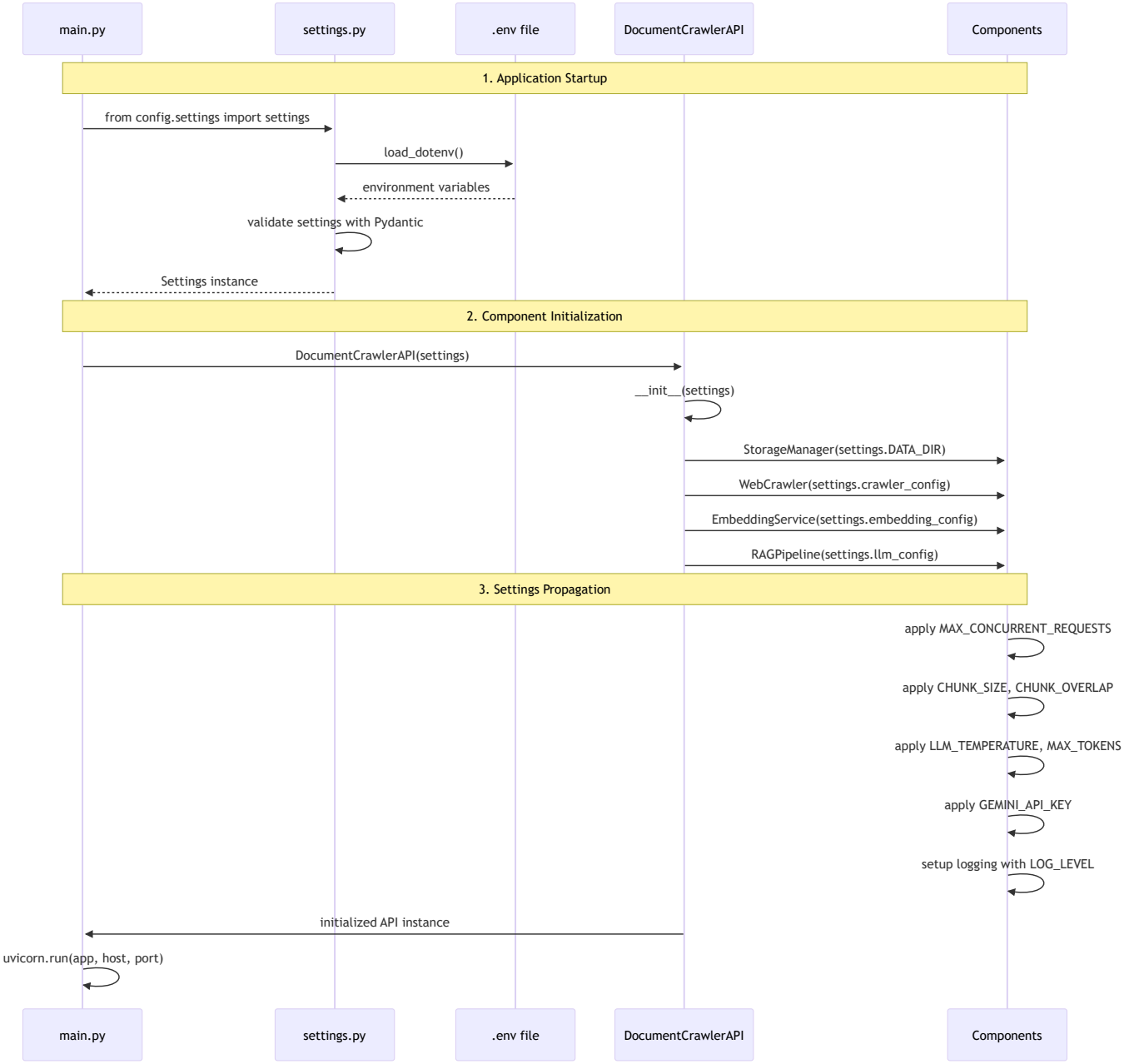

DocumentCrawlerAPI
BackgroundTask
EmbeddingService
StorageManager
VectorStore
GeminiAPI
SentenceTransformer
FileSystem
Client
1. Embedding Request
generate_embeddings()
validate EmbedRequest
validate domain exists
domain validated
create_background_task(_embed_task)
return TaskResponse
2. Document Loading
EmbeddingService(settings)
initialize google.generativeai
initialize SentenceTransformer
VectorStore(storage_manager)
process_documents(domain_name)
load_documents(domain_name, format="json")
read documents.json
document data
List[Document]
3. Document Chunking
loop
[For each document]
_chunk_document(document.content)
RecursiveCharacterTextSplitter(1000, 100)
split text into chunks (max 50)
create DocumentChunk objects
4. Embedding Generation
loop
[For each chunk]
generate_embedding(chunk.content)
try primary provider (Gemini)
genai.embed_content(chunk.content)
alt
[Gemini Success]
768-dim vector
L2 normalize vector
[Gemini Fail]
model.encode(chunk.content)
384-dim vector
L2 normalize vector
add embedding to chunk
5. Vector Store Creation
save_embeddings(chunks_with_embeddings)
_create_faiss_index(dimension)
create IndexFlatIP
_normalize_vectors(embeddings)
index.add(normalized_vectors)
_save_metadata(chunk_metadata)
write index.faiss
write metadata.json
write index_info.json
embedding_success
EmbeddingResult(success, count, dimension)
update background_tasks[task_id]

8.3 Complete Query Process Flow

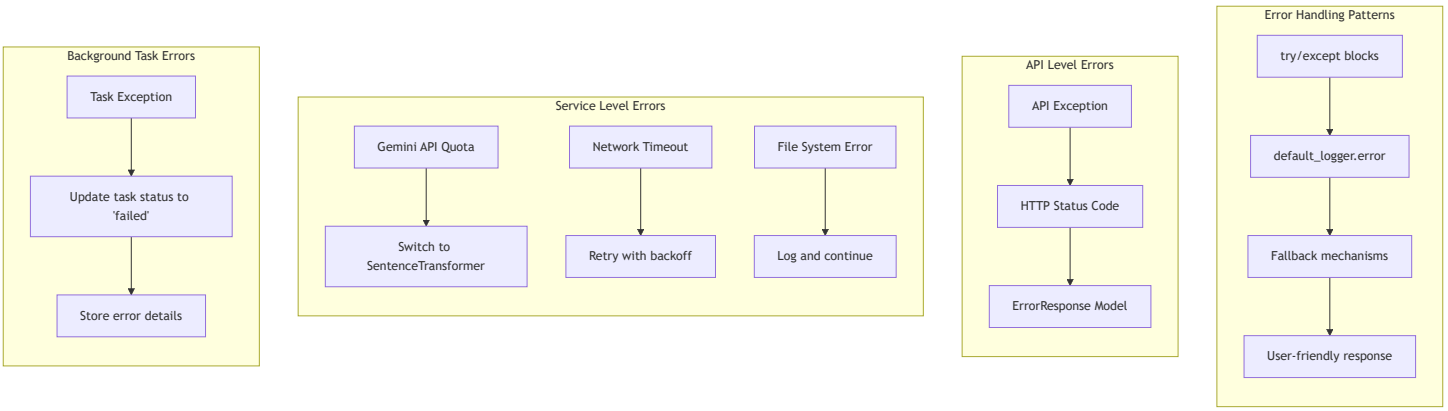

DocumentCrawlerAPI
RAGPipeline
EmbeddingService
VectorStore
GeminiAPI
StorageManager
Client
1. Query Request Processing
query_documentation()
validate QueryRequest
RAGPipeline(embedding_service, vector_store)
__init__ components
process_query(query, domain, top_k, include_context)
_validate_domain(domain)
check domain exists
domain_valid = True
2. Query Embedding Generation
generate_embedding(query)
genai.embed_content(query)
alt
[Gemini Success]
query_vector (768-dim)
[Gemini Fail]
sentence_transformer.encode(query)
query_vector (384-dim)
L2 normalize query_vector
normalized_query_vector
3. Vector Similarity Search
search(query_vector, top_k=5)
load_index(domain)
_normalize_vectors([query_vector])
index.search(query_vector, k=top_k)
get similarity scores and indices
_load_metadata() for result indices
filter by similarity threshold
create SearchResult objects
List[SearchResult]
4. Response Generation Decision
_check_llm_available() and include_context
alt
[LLM Available and Context Requested]
4a. LLM Response Generation
_generate_llm_answer(query, search_results)
build comprehensive prompt template
format context from search_results
assemble senior engineer prompt
genai.GenerativeModel.generate_content()
comprehensive_answer



8.4 Configuration Loading Flow



8.5 Error Handling Flow



9. Error Handling & Resilience

8.1 Error Handling Strategy

```
# Comprehensive Error Handling
ERROR_HANDLING = {
    "import_path_resolution": "Complex fallback import system for module loading",
    "api_failures": "Graceful degradation from Gemini to fallback embeddings/generation",
    "vector_store_errors": "Index validation and rebuilding capabilities",
    "crawler_failures": "URL validation, retry logic, failed URL tracking",
    "background_task_errors": "Exception capture with detailed error storage",
    "llm_failures": "Automatic fallback to structured response generation"
}
```

8.2 Logging Architecture

```
# Logging System
LOGGING_CONFIG = {
    "default_logger": "Centralized logging via utils.logger",
    "module_specific": "Per-module logger setup with configurable levels",
    "structured_logging": "Consistent format across all components",
    "error_tracking": "Detailed exception logging with context",
    "performance_metrics": "Query timing, confidence scores, processing statistics"
}
```

9. Performance & Scalability Considerations

9.1 Performance Optimizations

```
# Performance Features
PERFORMANCE_OPTIMIZATIONS = {
    "async_architecture": "Full async/await implementation throughout",
    "connection_pooling": "aiohttp session reuse for crawler",
    "vector_search": "FAISS for high-performance similarity search",
    "chunking_strategy": "Optimized chunk size for embedding efficiency",
    "background_processing": "Non-blocking crawl and embed operations",
    "caching": "Vector store persistence for reuse across sessions"
}
```

9.2 Scaling Strategies

```
# Scalability Considerations
SCALABILITY_FEATURES = {
    "domain_isolation": "Separate vector stores per domain",
    "concurrent_crawling": "Configurable request concurrency",
    "memory_management": "Efficient FAISS index loading/unloading",
    "stateless_design": "API endpoints designed for horizontal scaling",
    "background_task_tracking": "Distributed task management ready"
}
```

10. Development Workflow Integration

10.1 Development Patterns

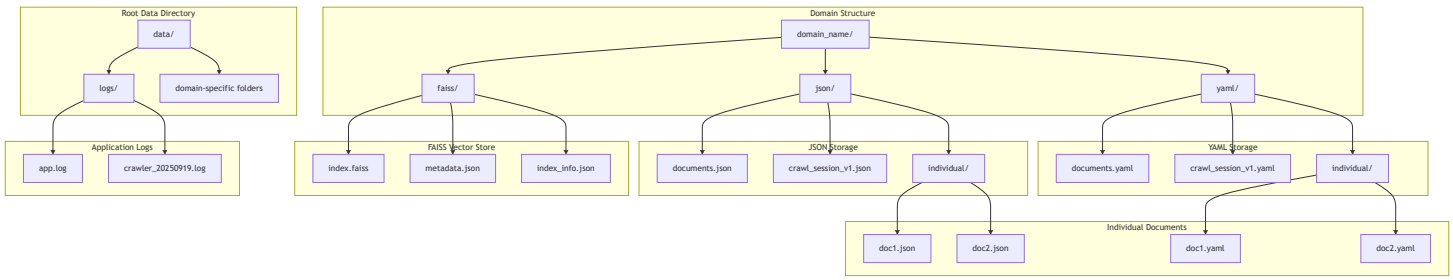
```
# Development Best Practices
DEV_PATTERNS = {
    "import_management": "Sophisticated fallback import system",
    "testing_workflow": "test_crawl.py for development validation",
    "configuration_management": ".env.example for setup guidance",
    "error_debugging": "Comprehensive logging for troubleshooting",
    "modular_architecture": "Clear separation of concerns across components"
}
```

10.2 Deployment Architecture

```
# Deployment Configuration
DEPLOYMENT_SETUP = {
    "entry_points": ["src/main.py", "run_server.py"],
    "environment_setup": "Virtual environment with requirements.txt",
    "configuration": ".env file with API keys and settings",
    "data_persistence": "Local file system storage for domains and vectors",
    "api_server": "FastAPI with Uvicorn for production deployment"
}
```

9. Domain-Based Storage Architecture

9.1 File System Organization



9.2 Storage Format Specifications

```
# Domain-Based Storage Structure
DOMAIN_STORAGE = {
    "domain_folder": {
        "json/": {
            "documents.json": "Complete document collection (JSON)",
            "crawl_session_v1.json": "Crawling session metadata",
            "individual/": {
                "{url_slug}.json": "Individual document files"
            }
        },
        "yaml/": {
            "documents.yaml": "Complete document collection (YAML)",
            "crawl_session_v1.yaml": "Crawling session metadata",
            "individual/": {
                "{url_slug}.yaml": "Individual document files"
            }
        },
        "faiss/": {
            "index.faiss": "FAISS vector index (IndexFlatIP)",
            "metadata.json": "Document chunk metadata and mappings",
            "index_info.json": "Index configuration and statistics"
        }
    }
}

# File Format Specifications
FILE_FORMATS = {
    "json_format": {
        "documents": "List[DocumentContent] - Full document objects",
        "chunks": "List[ContentChunk] - Embedding-ready chunks",
        "metadata": "Dict[str, Any] - Index and processing metadata"
    },
    "yaml_format": {
        "purpose": "Human-readable storage and inspection",
        "structure": "Same as JSON but YAML formatted",
        "usage": "Manual review and debugging"
    },
    "faiss_format": {
        "index_type": "IndexFlatIP for cosine similarity",
        "vector_dimensions": "768 (Gemini) or 384 (sentence-transformers)",
        "normalization": "L2 normalized vectors",
```

```
        "metric": "Inner product (cosine similarity)"
    }
}
```

10. Performance & Scalability Considerations

10.1 Performance Metrics

```
# Key Performance Indicators
PERFORMANCE_METRICS = {
    "crawling_performance": {
        "concurrent_requests": "10 simultaneous connections",
        "request_timeout": "30 seconds per URL",
        "retry_logic": "3 attempts with exponential backoff",
        "rate_limiting": "1 second delay between requests"
    },
    "embedding_performance": {
        "chunk_size": "1000 characters with 100 overlap",
        "max_chunks_per_doc": "50 chunks maximum",
        "batch_processing": "Sequential chunk processing",
        "vector_dimensions": "768d (Gemini) / 384d (fallback)"
    },
    "vector_search_performance": {
        "index_type": "FAISS IndexFlatIP",
        "similarity_metric": "Cosine similarity",
        "search_latency": "< 100ms for top-k retrieval",
        "memory_usage": "Efficient for domain-isolated indexes"
    },
    "llm_response_performance": {
        "generation_time": "< 2 seconds average",
        "temperature": "0.3 for consistent responses",
        "max_tokens": "4096 tokens per response",
        "fallback_mode": "Instant structured responses"
    }
}

# Optimization Strategies
OPTIMIZATIONS = {
    "async_architecture": "Full async/await implementation",
    "connection_pooling": "aiohttp session reuse",
    "vector_optimization": "FAISS for high-performance search",
    "memory_management": "Lazy loading and efficient structures",
    "background_processing": "Non-blocking operations"
}
```

10.2 Scalability Architecture

```
# Horizontal Scaling Options
SCALING_STRATEGIES = {
    "domain_isolation": "Separate vector stores per crawled domain",
    "concurrent_processing": "Configurable request concurrency limits",
    "stateless_api": "FastAPI endpoints ready for load balancing",
    "background_tasks": "Distributed task management ready",
    "storage_scaling": "File-based storage migrates to databases",
    "vector_db_scaling": "FAISS indexes per domain, load on demand"
}

# Resource Optimization
RESOURCE_OPTIMIZATION = {
    "memory_usage": {
        "vector_indexes": "Loaded on-demand per domain",
        "document_storage": "File-based with efficient serialization",
        "session_management": "aiohttp connection pooling"
    },
    "cpu_usage": {
        "embedding_generation": "Batch processing with fallbacks",
        "vector_search": "FAISS optimized operations",
        "content_processing": "Async processing pipelines"
    },
    "storage_optimization": {
        "dual_format": "JSON for processing, YAML for inspection",
        "compression": "Efficient serialization formats",
        "domain_partitioning": "Isolated storage per domain"
    }
}
```

11. Security Architecture

11.1 Security Layers

```
# Security Implementation
SECURITY_MEASURES = {
    "api_key_management": {
        "storage": "Environment variables only",
        "rotation": "Automatic fallback mechanisms",
        "exposure_prevention": "No logging of sensitive keys",
        "validation": "Pydantic settings validation"
    },
    "input_validation": {
        "url_validation": "URLUtils.is_valid_url() checks",
        "request_validation": "Pydantic models for all inputs",
        "domain_restriction": "Domain-based crawling restrictions",
        "content_sanitization": "trafilatura content extraction"
    },
    "error_handling": {
        "sensitive_data": "No API keys in error messages",
        "logging_safety": "Structured logging without secrets",
        "fallback_security": "Secure degradation without exposure",
        "rate_limiting": "Built-in concurrency controls"
    },
    "data_protection": {
        "file_permissions": "Standard OS file permissions",
        "data_isolation": "Domain-based data separation",
        "access_control": "Local file system security",
        "audit_trail": "Comprehensive operation logging"
    }
}
```

12. Deployment Architecture

12.1 Production Deployment Flow

12.2 Deployment Configuration

```python
# Production Deployment Setup
DEPLOYMENT_CONFIG = {
    "application": {
        "framework": "FastAPI with Uvicorn",
        "port": 8000,
        "workers": "Multiple Uvicorn workers",
        "host": "0.0.0.0 for container deployment"
    },
    "environment": {
        "python_version": "Python 3.12+",
        "virtual_environment": "venv or conda",
        "dependencies": "requirements.txt",
        "environment_variables": ".env file"
    },
    "data_persistence": {
        "storage_type": "Local file system",
        "domain_isolation": "Per-domain directories",
        "backup_strategy": "File system backups",
        "migration_path": "Ready for database migration"
    },
    "monitoring": {
        "health_checks": "/health endpoint",
        "logging": "Structured application logs",
        "performance": "Built-in timing metrics",
        "error_tracking": "Comprehensive error logging"
    }
}
```

13. Monitoring & Observability

13.1 Monitoring Stack

```
# Observability Components
MONITORING_ARCHITECTURE = {
    "application_logs": {
        "fastapi_logs": "Request/response logging with timing",
        "crawler_logs": "Crawling operation tracking",
        "embedding_logs": "Embedding generation monitoring",
        "rag_logs": "Query processing and response metrics",
        "error_logs": "Exception tracking with context"
    },
    "performance_metrics": {
        "api_performance": "Endpoint response times and throughput",
        "crawling_metrics": "URLs processed, success/failure rates",
        "embedding_metrics": "Processing time, chunk counts, dimensions",
        "vector_search_metrics": "Query latency, similarity scores",
        "llm_metrics": "Generation time, token usage, fallback rates"
    },
    "business_metrics": {
        "usage_analytics": "Query volume, domain popularity",
        "content_metrics": "Documents crawled, domains indexed",
        "quality_metrics": "Response confidence, user satisfaction",
        "system_health": "Background task success rates"
    },
    "infrastructure_metrics": {
        "resource_usage": "Memory, CPU, disk I/O",
        "external_api": "Gemini API usage and quotas",
        "storage_metrics": "Domain sizes, file counts",
        "concurrency": "Active connections and tasks"
    }
}
```

14. Future Architecture Enhancements

14.1 Recommended Improvements

```
# Architecture Evolution Roadmap
FUTURE_ENHANCEMENTS = {
    "distributed_processing": {
        "technology": "Redis/Celery for background tasks",
        "purpose": "Horizontal scaling of crawling and embedding",
        "impact": "Handle larger documentation sites concurrently"
    },
    "advanced_caching": {
        "technology": "Redis for response and vector caching",
        "purpose": "Reduce API costs and improve performance",
        "impact": "Faster responses for common queries"
    },
    "database_migration": {
        "technology": "PostgreSQL/MongoDB integration",
        "purpose": "Scalable storage beyond file system limits",
        "impact": "Support for larger document collections"
    },
    "real_time_features": {
        "technology": "WebSocket connections",
        "purpose": "Live crawling progress and status updates",
        "impact": "Better user experience for long-running tasks"
    },
    "advanced_rag": {
        "techniques": ["Query decomposition", "Multi-step reasoning", "Context ranking"],
        "purpose": "Improved answer quality and relevance",
        "impact": "More accurate and comprehensive responses"
    },
    "enterprise_security": {
        "technology": "OAuth2, RBAC, audit logging",
        "purpose": "Enterprise-grade security and compliance",
        "impact": "Production deployment in regulated environments"
    }
}
```

15. Comprehensive Architecture Summary

15.1 System Integration Overview

The AI-Powered Documentation Crawler represents a sophisticated, production-ready RAG system with the following architectural strengths:

Multi-Layer Architecture

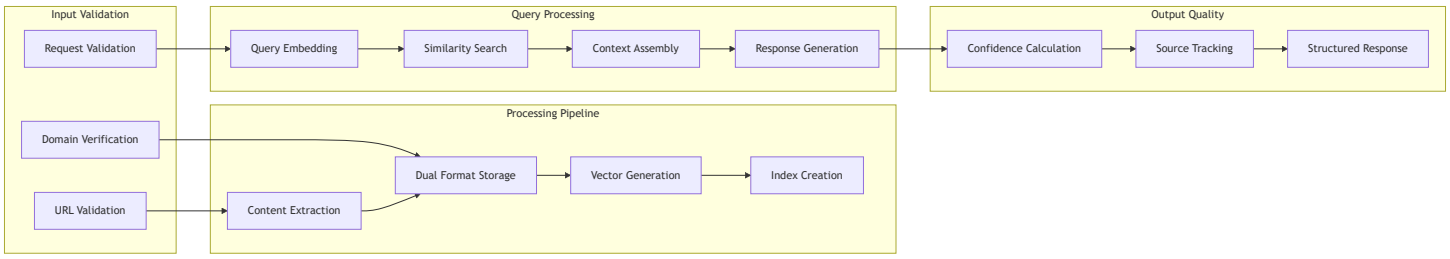
- **Presentation Layer**: FastAPI with OpenAPI documentation and async request handling
- **Application Services**: Background task management, request validation, and API orchestration
- **Core Business Logic**: Domain-specific services (Crawler, Embedding, RAG, Storage)
- **Infrastructure Services**: Configuration, logging, error handling, and health monitoring
- **Data Access Layer**: File system operations, vector store management, and metadata handling
- **External Integration**: Gemini API, sentence-transformers, and web content sources

Advanced RAG Capabilities

- **Dual Response Generation**: Comprehensive LLM-powered responses with intelligent structured fallbacks
- **Senior Engineer Prompt Engineering**: Production-ready prompt templates for actionable technical guidance
- **Multi-Provider Embedding**: Primary Gemini embeddings with sentence-transformer fallback
- **High-Performance Vector Search**: FAISS IndexFlatIP with cosine similarity and sub-second query response
- **Domain-Based Organization**: Isolated vector stores and document collections per crawled domain

Enterprise-Ready Features

- **Comprehensive Error Handling**: Multi-level fallback systems with graceful degradation
- **Async-First Architecture**: Full async/await implementation for maximum performance
- **Background Processing**: Non-blocking crawl and embedding operations with progress tracking
- **Dual-Format Storage**: Simultaneous JSON/YAML persistence for machine and human readability
- **Advanced Monitoring**: Structured logging, health checks, and performance metrics### 15.2 Data Flow Integrity



15.3 Operational Excellence

Development Workflow

- **Complex Import Resolution**: Sophisticated fallback import system for reliable module loading
- **Testing Framework**: `test_crawl.py` with comprehensive development validation workflows
- **Configuration Management**: Environment-based settings with intelligent defaults
- **Documentation**: Auto-generated OpenAPI docs with interactive testing interface

Production Readiness

- **Scalability**: Domain isolation for horizontal scaling, configurable concurrency controls
- **Reliability**: Multi-provider fallbacks, comprehensive error handling, and data consistency checks
- **Maintainability**: Clear separation of concerns, modular architecture, and extensive logging
- **Monitoring**: Health checks, performance metrics, and detailed operational visibility

Performance Characteristics

- **Crawling**: 10 concurrent requests with retry logic and rate limiting
- **Embedding**: Batch processing with 1000-character chunks and 100-character overlap
- **Vector Search**: Sub-second similarity search with FAISS optimization
- **Response Generation**: 0.3 temperature for consistent, professional responses

15.4 Future Enhancement Roadmap

The current architecture supports natural evolution toward:

Horizontal Scaling

- **Distributed Processing**: Background tasks ready for Redis/Celery integration
- **Load Balancing**: Stateless API design supports multiple instance deployment
- **Database Migration**: File-based storage can migrate to PostgreSQL/MongoDB
- **Caching Layer**: Redis integration for vector store and response caching

Advanced Features

- **Multi-Modal Support**: Architecture supports image and document processing extensions
- **Real-Time Updates**: WebSocket integration for live crawling progress
- **Advanced Analytics**: Query pattern analysis and content recommendation systems
- **Enterprise Security**: OAuth2, RBAC, and audit logging integration points

AI/ML Enhancements

- **Model Fine-Tuning**: Custom embedding models for domain-specific optimization
- **Advanced RAG**: Hypothetical document embeddings, multi-step reasoning
- **Quality Scoring**: Automated content quality assessment and filtering
- **Semantic Routing**: Dynamic model selection based on query complexity

15.5 Technical Specifications Summary

```
# Complete System Specifications
SYSTEM_SPECS = {
    "architecture_type": "Microservices-oriented with clear service boundaries",
    "api_framework": "FastAPI with async/await throughout",
    "storage_pattern": "Domain-based with JSON/YAML dual format",
    "vector_database": "FAISS with IndexFlatIP for cosine similarity",
    "embedding_models": "Gemini-001 (768d) with sentence-transformers fallback (384d)",
    "llm_integration": "Gemini 1.5 Flash with comprehensive prompt engineering",
    "concurrency_model": "AsyncIO with semaphore-based request limiting",
    "error_handling": "Multi-level fallbacks with graceful degradation",
    "monitoring": "Structured logging with performance metrics",
    "configuration": "Environment-based with Pydantic validation",
    "testing": "Comprehensive development validation framework",
    "documentation": "Auto-generated OpenAPI with interactive interface"
}
```

This comprehensive micro-level architecture documentation provides both the high-level understanding and detailed technical specifications necessary for developers, architects, and operations teams to effectively work with, maintain, and extend the AI-Powered Documentation Crawler system.