**Project Citadel: Comprehensive Mermaid Diagrams**

**1. Runtime Architecture Diagram**

*Who talks to whom, when, and how*

mermaid

graph TB

subgraph "User Interface Layer"

CLI[Command Line Interface]

STREAM[Streamlit Web Interface]

end

subgraph "Application Layer"

MAIN[insert\_docs.py<br/>Main Orchestrator]

RAG[rag\_agent.py<br/>AI Agent]

UTILS[utils.py<br/>DB Utilities]

end

subgraph "Crawling Engine"

C1[1-crawl\_single\_page.py]

C2[2-crawl\_docs\_sequential.py]

C3[3-crawl\_sitemap\_parallel.py]

C4[4-crawl\_llms\_txt.py]

C5[5-crawl\_site\_recursive.py]

CRAWL4AI[Crawl4AI Framework]

BROWSER[Playwright Browser]

end

subgraph "Processing Layer"

CHUNK[Hierarchical Chunker]

META[Metadata Extractor]

EMBED[Embedding Generator]

end

subgraph "Storage Layer"

CHROMA[(ChromaDB Vector Store)]

PERSIST[(Persistent Storage)]

end

subgraph "AI Services"

OPENAI[OpenAI API]

MODELS[LLM Models]

VECTORS[Vector Embeddings]

end

%% Runtime Relationships - Ingestion Flow

CLI -->|1. Command Args| MAIN

MAIN -->|2. URL Type Detection| MAIN

MAIN -->|3a. Regular Site| C5

MAIN -->|3b. Sitemap| C3

MAIN -->|3c. Text/Markdown| C4

MAIN -->|3d. Sequential| C2

MAIN -->|3e. Single Page| C1

C1 & C2 & C3 & C4 & C5 -->|4. HTTP Requests| CRAWL4AI

CRAWL4AI -->|5. Browser Control| BROWSER

BROWSER -->|6. Web Content| CRAWL4AI

CRAWL4AI -->|7. Markdown Content| MAIN

MAIN -->|8. Content Processing| CHUNK

CHUNK -->|9. Text Chunks| META

META -->|10. Metadata + Chunks| EMBED

EMBED -->|11. Vector Generation| OPENAI

OPENAI -->|12. Embeddings| EMBED

EMBED -->|13. Vectors + Metadata| UTILS

UTILS -->|14. Batch Operations| CHROMA

CHROMA -->|15. Persistence| PERSIST

%% Runtime Relationships - Query Flow

STREAM -->|Q1. User Question| RAG

RAG -->|Q2. Query Processing| RAG

RAG -->|Q3. Vector Search| UTILS

UTILS -->|Q4. Similarity Query| CHROMA

CHROMA -->|Q5. Relevant Chunks| UTILS

UTILS -->|Q6. Formatted Context| RAG

RAG -->|Q7. LLM Prompt| OPENAI

OPENAI -->|Q8. Generated Response| RAG

RAG -->|Q9. Final Answer| STREAM

%% Styling

classDef userLayer fill:#e1f5fe

classDef appLayer fill:#f3e5f5

classDef crawlLayer fill:#e8f5e8

classDef processLayer fill:#fff3e0

classDef storageLayer fill:#fce4ec

classDef aiLayer fill:#f1f8e9

class CLI,STREAM userLayer

class MAIN,RAG,UTILS appLayer

class C1,C2,C3,C4,C5,CRAWL4AI,BROWSER crawlLayer

class CHUNK,META,EMBED processLayer

class CHROMA,PERSIST storageLayer

class OPENAI,MODELS,VECTORS aiLayer

**2. Data Flow Diagram - Complete Ingestion + Query Pipeline**

mermaid

flowchart TD

subgraph "INGESTION PIPELINE"

START([URL Input]) --> DETECT{Content Type Detection}

DETECT -->|Regular Website| REC[Recursive Crawler<br/>5-crawl\_site\_recursively.py]

DETECT -->|Sitemap XML| PAR[Parallel Crawler<br/>3-crawl\_sitemap\_parallel.py]

DETECT -->|Text/Markdown| TXT[Text Crawler<br/>4-crawl\_llms\_txt.py]

DETECT -->|Single Page| SEQ[Sequential Crawler<br/>2-crawl\_docs\_sequential.py]

REC --> EXTRACT[Content Extraction<br/>HTML → Markdown]

PAR --> EXTRACT

TXT --> EXTRACT

SEQ --> EXTRACT

EXTRACT --> CHUNK[Hierarchical Chunking<br/>H1 → H2 → H3 → Char]

CHUNK --> METADATA[Metadata Extraction<br/>Headers, Counts, URLs]

METADATA --> EMBEDDING[Vector Embedding<br/>SentenceTransformers]

EMBEDDING --> BATCH[Batch Processing<br/>ChromaDB Insertion]

BATCH --> STORE[(Vector Database<br/>ChromaDB)]

end

subgraph "QUERY PIPELINE"

QUERY([User Question]) --> AGENT[RAG Agent<br/>Pydantic AI]

AGENT --> ANALYZE[Query Analysis<br/>Search Strategy]

ANALYZE --> SEARCH[Vector Similarity<br/>Search]

SEARCH --> RETRIEVE[Document Retrieval<br/>Top-K Results]

RETRIEVE --> CONTEXT[Context Formatting<br/>Relevance Scoring]

CONTEXT --> LLM[LLM Processing<br/>OpenAI API]

LLM --> RESPONSE[Response Generation<br/>Source Attribution]

RESPONSE --> OUTPUT([Final Answer])

end

subgraph "CROSS-PIPELINE DATA FLOW"

STORE -.->|Vector Similarity| SEARCH

CONTEXT -.->|Retrieved Context| LLM

end

subgraph "RESOURCE MANAGEMENT"

MEM[Memory Monitor<br/>PSUtil Tracking] -.->|Resource Control| PAR

MEM -.->|Adaptive Batching| REC

DISP[Adaptive Dispatcher<br/>70% Memory Threshold] -.->|Session Control| PAR

DISP -.->|Concurrency Limits| REC

end

%% Data Types and Formats

EXTRACT -.->|Markdown Text| CHUNK

CHUNK -.->|Text Chunks[]| METADATA

METADATA -.->|{chunk, meta}[]| EMBEDDING

EMBEDDING -.->|{vectors, ids, meta}[]| BATCH

SEARCH -.->|{docs, distances, ids}| RETRIEVE

RETRIEVE -.->|Ranked Results| CONTEXT

CONTEXT -.->|Formatted Context String| LLM

%% Styling

classDef ingestion fill:#e8f5e8

classDef query fill:#e1f5fe

classDef storage fill:#fce4ec

classDef resource fill:#fff3e0

class START,DETECT,REC,PAR,TXT,SEQ,EXTRACT,CHUNK,METADATA,EMBEDDING,BATCH ingestion

class QUERY,AGENT,ANALYZE,SEARCH,RETRIEVE,CONTEXT,LLM,RESPONSE,OUTPUT query

class STORE storage

class MEM,DISP resource

**3. System Component Architecture**

mermaid

graph TB

subgraph "PRESENTATION TIER"

CLI[Command Line Interface<br/>insert\_docs.py entry point]

WEB[Streamlit Web Interface<br/>streamlit\_app.py]

end

subgraph "APPLICATION TIER"

subgraph "Core Orchestration"

MAIN[Main Orchestrator<br/>insert\_docs.py]

RAG[RAG Agent<br/>rag\_agent.py]

UTIL[Database Utilities<br/>utils.py]

end

subgraph "Crawler Modules"

SINGLE[Single Page<br/>1-crawl\_single\_page.py]

SEQ[Sequential<br/>2-crawl\_docs\_sequential.py]

PARALLEL[Parallel Sitemap<br/>3-crawl\_sitemap\_parallel.py]

MARKDOWN[Markdown/Text<br/>4-crawl\_llms\_txt.py]

RECURSIVE[Recursive Site<br/>5-crawl\_site\_recursive.py]

end

subgraph "Processing Components"

DETECTOR[URL Type Detector]

CHUNKER[Hierarchical Chunker]

EXTRACTOR[Metadata Extractor]

EMBEDDER[Vector Embedder]

end

end

subgraph "INTEGRATION TIER"

subgraph "Web Crawling"

CRAWL4AI[Crawl4AI Framework]

PLAYWRIGHT[Playwright Browser Engine]

DISPATCHER[Memory Adaptive Dispatcher]

end

subgraph "AI/ML Services"

OPENAI\_API[OpenAI API Client]

SENTENCE\_T[SentenceTransformers]

PYDANTIC\_AI[Pydantic AI Agent Framework]

end

end

subgraph "DATA TIER"

CHROMA[(ChromaDB Vector Database)]

FS[(File System<br/>Persistent Storage)]

CONFIG[(Configuration<br/>.env files)]

end

%% Component Interactions

CLI --> MAIN

WEB --> RAG

MAIN --> DETECTOR

MAIN --> CHUNKER

MAIN --> EXTRACTOR

MAIN --> EMBEDDER

MAIN --> UTIL

RAG --> UTIL

RAG --> PYDANTIC\_AI

DETECTOR --> SINGLE

DETECTOR --> SEQ

DETECTOR --> PARALLEL

DETECTOR --> MARKDOWN

DETECTOR --> RECURSIVE

SINGLE & SEQ & PARALLEL & MARKDOWN & RECURSIVE --> CRAWL4AI

CRAWL4AI --> PLAYWRIGHT

PARALLEL & RECURSIVE --> DISPATCHER

EMBEDDER --> SENTENCE\_T

RAG --> OPENAI\_API

UTIL --> CHROMA

CHROMA --> FS

MAIN --> CONFIG

RAG --> CONFIG

%% Styling

classDef presentation fill:#e1f5fe

classDef application fill:#f3e5f5

classDef integration fill:#e8f5e8

classDef data fill:#fce4ec

class CLI,WEB presentation

class MAIN,RAG,UTIL,SINGLE,SEQ,PARALLEL,MARKDOWN,RECURSIVE,DETECTOR,CHUNKER,EXTRACTOR,EMBEDDER application

class CRAWL4AI,PLAYWRIGHT,DISPATCHER,OPENAI\_API,SENTENCE\_T,PYDANTIC\_AI integration

class CHROMA,FS,CONFIG data

**4. Detailed Sequence Diagram - Complete Workflow**

mermaid

sequenceDiagram

participant User

participant CLI as insert\_docs.py

participant Detector as URL Detector

participant Crawler as Crawler Module

participant Crawl4AI

participant Browser as Playwright

participant Processor as Content Processor

participant Embedder as Vector Embedder

participant OpenAI as OpenAI API

participant Utils as utils.py

participant ChromaDB

participant RAGAgent as rag\_agent.py

participant WebUI as Streamlit UI

%% INGESTION SEQUENCE

rect rgb(232, 245, 232)

Note over User,ChromaDB: DOCUMENT INGESTION PIPELINE

User->>CLI: python insert\_docs.py <URL> [options]

CLI->>CLI: Parse command arguments

CLI->>Detector: detect\_url\_type(url)

alt Regular Website

Detector->>Crawler: select\_recursive\_crawler()

else Sitemap XML

Detector->>Crawler: select\_parallel\_crawler()

else Text/Markdown

Detector->>Crawler: select\_text\_crawler()

end

Crawler->>Crawl4AI: initialize\_session(config)

Crawl4AI->>Browser: launch\_browser(headless=True)

Browser-->>Crawl4AI: browser\_session

loop For each URL batch

Crawler->>Crawl4AI: arun\_many(urls, dispatcher)

Crawl4AI->>Browser: navigate\_and\_extract(url)

Browser-->>Crawl4AI: html\_content

Crawl4AI->>Crawl4AI: convert\_to\_markdown()

Crawl4AI-->>Crawler: {url, markdown, links}

end

Crawler-->>CLI: crawled\_content[]

CLI->>Processor: smart\_chunk\_markdown(content)

Processor->>Processor: hierarchical\_split(H1→H2→H3→char)

Processor-->>CLI: text\_chunks[]

CLI->>Processor: extract\_metadata(chunks)

Processor-->>CLI: {chunks, metadata}[]

CLI->>Embedder: generate\_embeddings(chunks)

Embedder->>OpenAI: create\_embeddings(text[])

OpenAI-->>Embedder: vector\_embeddings[]

Embedder-->>CLI: {vectors, chunks, metadata}

CLI->>Utils: add\_documents\_to\_collection()

Utils->>ChromaDB: batch\_insert(vectors, metadata)

ChromaDB-->>Utils: success\_confirmation

Utils-->>CLI: insertion\_complete

CLI-->>User: "Successfully inserted N documents"

end

%% QUERY SEQUENCE

rect rgb(225, 245, 254)

Note over User,WebUI: QUERY & RESPONSE PIPELINE

User->>WebUI: "Ask question about docs"

WebUI->>RAGAgent: process\_question(query)

RAGAgent->>RAGAgent: analyze\_question()

RAGAgent->>Utils: retrieve(search\_query, n\_results=5)

Utils->>ChromaDB: query\_collection(query\_embedding)

ChromaDB->>ChromaDB: vector\_similarity\_search()

ChromaDB-->>Utils: {docs, distances, metadata}

Utils->>Utils: format\_results\_as\_context()

Utils-->>RAGAgent: formatted\_context\_string

RAGAgent->>OpenAI: chat\_completion(context + question)

OpenAI-->>RAGAgent: generated\_response

RAGAgent->>RAGAgent: add\_source\_attribution()

RAGAgent-->>WebUI: final\_answer\_with\_sources

WebUI-->>User: Display response with references

end

%% RESOURCE MANAGEMENT

rect rgb(255, 243, 224)

Note over Crawler,Browser: RESOURCE MANAGEMENT (Parallel)

Crawler->>Crawler: monitor\_memory\_usage()

alt Memory > 70%

Crawler->>Crawler: reduce\_concurrent\_sessions()

Crawler->>Browser: close\_excess\_sessions()

else Memory < 50%

Crawler->>Crawler: increase\_concurrent\_sessions()

Crawler->>Browser: open\_additional\_sessions()

end

end

**5. Data Structure Flow Diagram**

mermaid

flowchart LR

subgraph "INPUT FORMATS"

URL1[Website URL<br/>https://docs.site.com]

URL2[Sitemap URL<br/>https://site.com/sitemap.xml]

URL3[Text File URL<br/>https://site.com/docs.txt]

end

subgraph "CRAWLING OUTPUT"

RAW[Raw Content<br/>{url: string,<br/>markdown: string,<br/>links: array}]

end

subgraph "PROCESSING STAGES"

CHUNKS[Text Chunks<br/>[string, string, ...]]

META[Metadata Objects<br/>[{headers: array,<br/>char\_count: int,<br/>word\_count: int,<br/>source\_url: string}, ...]]

VECTORS[Vector Objects<br/>[{embedding: float[],<br/>text: string,<br/>metadata: object}, ...]]

end

subgraph "STORAGE FORMAT"

CHROMA\_DOC[ChromaDB Documents<br/>{ids: [string],<br/>documents: [string],<br/>metadatas: [object],<br/>embeddings: [float[]]}]

end

subgraph "QUERY PROCESSING"

QUERY\_VEC[Query Vector<br/>float[384]]

RESULTS[Search Results<br/>{documents: [string],<br/>distances: [float],<br/>metadatas: [object],<br/>ids: [string]}]

CONTEXT[Formatted Context<br/>"Document 1 (relevance: 0.85):<br/>content...<br/><br/>Document 2 (relevance: 0.72):<br/>content..."]

end

subgraph "AI PROCESSING"

PROMPT[LLM Prompt<br/>"Context: [formatted\_context]<br/><br/>Question: [user\_question]<br/><br/>Answer based on context:"]

RESPONSE[Final Response<br/>"Based on the documentation,<br/>the answer is...<br/><br/>Sources: doc1.html, doc2.html"]

end

%% Data Flow

URL1 & URL2 & URL3 --> RAW

RAW --> CHUNKS

CHUNKS --> META

META --> VECTORS

VECTORS --> CHROMA\_DOC

CHROMA\_DOC -.->|Query Time| QUERY\_VEC

QUERY\_VEC --> RESULTS

RESULTS --> CONTEXT

CONTEXT --> PROMPT

PROMPT --> RESPONSE

%% Data Transformations

RAW -.->|smart\_chunk\_markdown()| CHUNKS

CHUNKS -.->|extract\_section\_info()| META

META -.->|SentenceTransformers| VECTORS

VECTORS -.->|batch\_insert()| CHROMA\_DOC

RESULTS -.->|format\_results\_as\_context()| CONTEXT

CONTEXT -.->|Template + Question| PROMPT

PROMPT -.->|OpenAI API| RESPONSE

%% Styling

classDef input fill:#e8f5e8

classDef processing fill:#fff3e0

classDef storage fill:#fce4ec

classDef query fill:#e1f5fe

classDef ai fill:#f1f8e9

class URL1,URL2,URL3 input

class RAW,CHUNKS,META,VECTORS processing

class CHROMA\_DOC storage

class QUERY\_VEC,RESULTS,CONTEXT query

class PROMPT,RESPONSE ai

**6. Memory and Resource Management Flow**

mermaid

stateDiagram-v2

[\*] --> Initializing

Initializing --> MemoryCheck : Start Crawling

state MemoryCheck {

[\*] --> CheckMemory

CheckMemory --> LowMemory : < 50%

CheckMemory --> NormalMemory : 50-70%

CheckMemory --> HighMemory : > 70%

LowMemory --> IncreaseThreads : Scale Up

NormalMemory --> MaintainThreads : Stable

HighMemory --> DecreaseThreads : Scale Down

IncreaseThreads --> CheckMemory : Monitor

MaintainThreads --> CheckMemory : Monitor

DecreaseThreads --> CheckMemory : Monitor

}

state CrawlingOperation {

[\*] --> BatchDispatch

BatchDispatch --> ParallelCrawl : Dispatch URLs

ParallelCrawl --> SessionManagement

SessionManagement --> ContentExtraction

ContentExtraction --> ResultAggregation

ResultAggregation --> [\*]

SessionManagement --> SessionReuse : Reuse Available

SessionManagement --> NewSession : Create New

SessionReuse --> ContentExtraction

NewSession --> ContentExtraction

}

MemoryCheck --> CrawlingOperation : Resource Allocated

CrawlingOperation --> MemoryCheck : Batch Complete

state ErrorHandling {

[\*] --> DetectError

DetectError --> MemoryExhaustion : OOM Error

DetectError --> NetworkError : Connection Failed

DetectError --> TimeoutError : Request Timeout

MemoryExhaustion --> ReduceLoad : Scale Down

NetworkError --> RetryRequest : Retry Logic

TimeoutError --> IncreaseTimeout : Adjust Config

ReduceLoad --> [\*] : Resume

RetryRequest --> [\*] : Resume

IncreaseTimeout --> [\*] : Resume

}

CrawlingOperation --> ErrorHandling : Error Detected

ErrorHandling --> MemoryCheck : Error Resolved

MemoryCheck --> Completed : All URLs Processed

Completed --> [\*]

These comprehensive diagrams provide a complete technical foundation for understanding Project Citadel's architecture, data flows, and runtime behavior, supporting both development and stakeholder communication needs.