Python Codebase Schematic Document

Project Overview

This document provides a comprehensive schematic of the Python codebase showing file hierarchy, classes, functions, and methods organized by file.

app.py



Main Flask Application: Central application file implementing two-track architecture with Chain Operations and Enhanced Chat separation.

Key Functions:

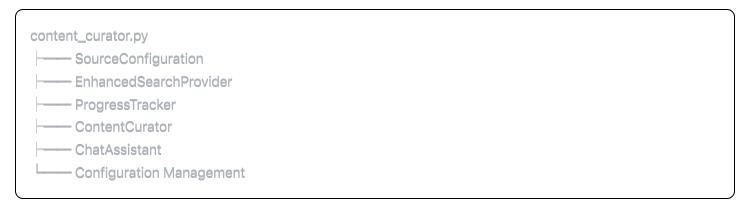
- (index()) Serves main HTML template
- (new_page()) Serves architecture page template
- (research_agent_search()) Internet search endpoint for research agent
- (research_agent_search_progress()) Get search progress for sessions
- (get_frontend_messages()) Get queued messages for frontend display
- (download_file()) Enhanced file operations endpoint with directory routing
- (get_interests()) Get all interests from database
- (interest_control()) Handle interest management control actions
- (customer_control()) Customer management control endpoint
- (toggle_starred()) Toggle starred status of articles
- (check_pdf_capabilities()) Check PDF parsing capabilities
- (verify_architecture()) Verify two-track architecture setup

Error Handlers:

• (not_found()) - 404 error handler

• (internal_error()) - 500 error handler

content_curator.py



Content Curation System: Handles RSS feeds, arXiv queries, article collection, and Al-powered content processing.

Class: SourceConfiguration

- (__init__(config_file: str))
- (load_config()) Load configuration from JSON file
- (get_rss_feeds(category: Optional[str])) Get RSS feed URLs
- (get_arxiv_queries()) Get arXiv search queries
- (get_arxiv_categories()) Get arXiv categories
- (get_setting(key: str, default)) Get configuration setting
- (reload_config()) Reload configuration from file

Class: EnhancedSearchProvider

- (__init__())
- (search(query: str, max_results: int, provider: str)) Search using specified provider
- (_search_duckduckgo(query: str, max_results: int)) DuckDuckGo search implementation

Class: ProgressTracker

- (__init__()
- (update_phase(phase: str)) Update current operation phase
- (set_message(message: str)) Set progress message
- (get_status()) Get current status

Class: ContentCurator

- __init__(db_path: str, config_file: str)
- (setup_database()) Initialize SQLite database tables
- (collect_arxiv_papers(query: str, max_results: int)) Collect papers from arXiv
- (collect_rss_feeds(feeds: List[str], topic_filter: str)) Collect RSS feed content
- (process_content_with_ai(content_batch: List[Dict])) Al processing of content
- (search_all_sources(topic: str, max_results_per_source: int)) Search across all sources
- (run_collection_cycle(topic: str, max_process: int)) Run complete collection cycle
- (prune_old_content()) Remove old content from database

Class: ChatAssistant

- (__init__(ollama_model: str))
- (generate_summary(content: str, context: str)) Generate Al summary
- (calculate_relevance(content: str, interests: List[str])) Calculate content relevance
- (chat_with_database(message: str, session_id: str)) Chat interface with database

research_agent.py

```
research_agent.py

----- SessionManager

----- ResearchSession

----- ResearchAssistant

----- Factory Functions
```

Research Assistant System: Simplified coordination layer for session management and enhanced chat features.

Class: SessionManager

- __init__(db_path: str)
- (get_or_create_session(session_id: str)) Get or create research session
- (cleanup_old_sessions()) Remove expired sessions
- (get_active_sessions()) Get list of active sessions

Class: ResearchSession

- (__init__(session_id: str, db_path: str))
- (update_activity()) Update last activity timestamp
- (get_context(key: str, default)) Get context value
- (set_context(key: str, value)) Set context value
- (to_dict()) Convert session to dictionary

Class: ResearchAssistant

- (__init__(content_curator, chat_assistant, data_directory: str))
- (get_or_create_session(session_id: str)) Get or create session
- (enhanced_chat_with_context(message: str, session_id: str)) Enhanced chat with context
- (chat_with_database(message: str, session_id: str)) Basic database chat
- (get_session_status(session_id: str)) Get session status and context

Factory Functions:

(create_research_agent_with_enhanced_features()) - Create research agent instance

customer_insights.py

```
customer_insights.py

— CustomerDataManager

— CustomerInsightsAgent

— ContextAwareAnalyzer

— BattleCardGenerator

L Analysis Pipeline
```

Customer Analysis System: Analyzes individual customers vs customer clusters with scenario-based processing.

Class: CustomerDataManager

- __init__(data_directory: str)
- (_load_customers()) Load customer CSV data
- (_load_company_data()) Load company information from JSON
- (get_customer_data()) Get customer data with caching
- (get_clusters()) Get unique cluster names

• (find_customers(query: str)) - Find customers by name or cluster

Class: CustomerInsightsAgent

- __init__(data_manager, cache_manager, ollama_model: str)
- (analyze_customer(customer_input: str)) Analyze individual customer
- (analyze_cluster(cluster_name: str)) Analyze customer cluster
- (generate_insights_report(customer_data: Dict)) Generate comprehensive insights
- (create_battle_card(customer_name: str)) Create customer battle card
- (_determine_analysis_type(customer_input: str)) Determine analysis approach

Class: ContextAwareAnalyzer

- (__init__(ollama_model: str))
- (analyze_content_for_customer(content: List[Dict], customer_context: Dict)) Content analysis
- (generate_strategic_insights(research_data: List[Dict])) Strategic insights generation
- (create_executive_summary(insights: Dict)) Executive summary creation

Class: BattleCardGenerator

- (__init__(company_data: Dict, ollama_model: str))
- (generate_battle_card(customer_data: Dict, research_data: List[Dict])) Generate battle card
- (_format_battle_card_sections(sections: Dict)) Format battle card sections

customer_mgmt.py

customer_mgmt.py

—— CustomerManager

—— Workflow Management

—— CSV Operations

L—— Integration Components

Customer Management Module: Handles customer creation, modification, and deletion with multistep workflows.

Class: CustomerManager

- __init__(data_directory: str, cache_directory: str)
- (is_customer_command(message: str)) Check if message is customer command

- (process_command_with_session(message: str, session_id: str)) Process command with session
- (_load_customers_csv()) Load customers CSV file
- (_save_customers_csv(df: pd.DataFrame)) Save customers CSV file
- (list_customers()) List all customers
- (start_add_customer_workflow(session_id: str)) Start customer addition workflow
- _add_customer_to_csv(customer_data: Dict) Add customer to CSV
- (get_help()) Get help information
- (cleanup_expired_sessions(timeout_seconds: int)) Clean up expired sessions

Factory Functions:

- (create_customer_manager()) Create CustomerManager instance
- (process_customer_command()) Process natural language customer command

cache_mgmt.py

```
cache_mgmt.py

CustomerCacheManager

File Processing Methods

Content Extraction

File Type Handlers
```

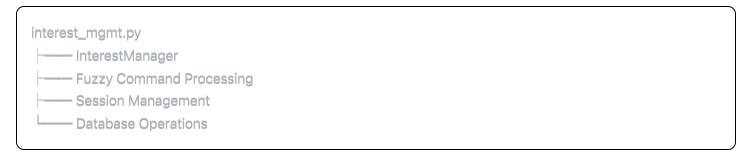
Customer Cache Management: Handles local file caching and content extraction for customer insights.

Class: CustomerCacheManager

- __init__(data_directory: str, cache_directory: str)
- (get_customer_research(customer_input: str)) Main entry point for customer research
- (_load_customers_csv()) Load customers CSV file
- (_find_customers(customer_input: str, customers_df: pd.DataFrame)) Find matching customers
- (_load_cached_files(customer_name: str)) Load cached files for customer
- (_extract_content_by_type(file_path: str)) Central dispatcher for file types
- (_extract_pdf_content(file_path: str)) Extract text from PDF files
- (_extract_html_content(file_path: str)) Extract text from HTML files

- (_extract_md_content(file_path: str)) Extract text from Markdown files
- (_extract_docx_content(file_path: str)) Extract text from Word documents
- (_extract_xlsx_content(file_path: str)) Extract text from Excel spreadsheets
- (_extract_pptx_content(file_path: str)) Extract text from PowerPoint presentations
- (_truncate_content(content: str)) Truncate content to max length

interest_mgmt.py



Interest Management System: Manages user interests with natural language processing and database persistence.

Class: InterestManager

- __init__(db_path: str, ollama_model: str)
- (is_interests_command(message: str)) Check if message is interest command
- (process_command_with_session(message: str, session_id: str)) Process with session tracking
- (get_all_interests()) Get all interests from database
- (add_interest(topic: str, weight: float)) Add new interest
- (remove_interest(interest_id: int)) Remove interest by ID
- (update_interest_weight(interest_id: int, new_weight: float)) Update interest weight
- (search_interests(query: str)) Search interests by query
- (get_help()) Get help information
- (cleanup_expired_sessions(timeout_seconds: int)) Clean up expired sessions

Factory Functions:

- (create_interest_manager()) Create InterestManager instance
- (process_interest_command()) Process natural language interest command

fuzzy_commands.py

•
fuzzy_commands.py
——— DynamicFuzzyPatternGenerator
Pattern Generation
Entity Extraction
Command Detection

Dynamic Fuzzy Logic Pattern Generator: Generates fuzzy logic patterns directly from customers.csv data.

Class: DynamicFuzzyPatternGenerator

- (__init__(customers_csv_path: str))
- (_load_csv_data()) Load CSV data and check modifications
- (_extract_keywords_from_text(text: str)) Extract keywords from text
- (generate_dynamic_patterns()) Generate patterns from CSV data
- (get_cached_patterns()) Get cached patterns with auto-refresh
- (_generate_customer_patterns()) Generate customer-specific patterns
- (_generate_cluster_patterns()) Generate cluster-specific patterns

Class: DynamicPattern

 Dataclass representing fuzzy pattern with name, exact patterns, fuzzy keywords, entity data, and source column

Functions:

 detect_fuzzy_command(message: str, customers_csv_path: str)) - Detect fuzzy commands in messages

entity_resolver.py

entity_resolver.py	
EntityExtractor	
CustomerEntityResolver	
IntentEntityResolver	
DisambiguationManager	
EntityResolutionPipeline	
L Utility Functions	

Entity Resolution System: Resolves customer names, clusters, and intents with fuzzy matching and disambiguation.

Class: EntityExtractor

- (__init__()
- (extract_entities(text: str)) Extract potential entity mentions from text
- (_extract_with_patterns(text: str)) Extract entities using regex patterns

Class: CustomerEntityResolver

- (__init__(data_directory: str))
- (resolve_customer_entity(mention)) Resolve customer entity mentions
- _find_exact_matches() Find exact matches
- (_find_fuzzy_matches()) Find fuzzy matches using similarity
- (_find_abbreviation_matches()) Find abbreviation/initials matches
- (_find_partial_matches()) Find partial substring matches
- (_rank_and_filter_candidates()) Rank and filter candidate matches

Class: IntentEntityResolver

- (__init__()
- (resolve_intent(text: str)) Resolve user intent from text
- (_calculate_intent_confidence()) Calculate confidence in intent resolution
- (_find_fuzzy_intent_matches()) Find fuzzy matches for intent keywords

Class: DisambiguationManager

- (__init__()
- (create_disambiguation_prompt()) Create disambiguation prompt for users
- _create_multiple_choice_prompt()) Create multiple choice prompt

Class: EntityResolutionPipeline

- (__init__(data_directory: str))
- (resolve_message(message: str)) Main entry point for entity resolution
- (_calculate_overall_confidence()) Calculate overall confidence in resolution

Utility Functions:

- (normalize_text(text: str)) Normalize text for comparison
- (calculate_similarity(text1: str, text2: str)) Calculate similarity between texts
- (extract_abbreviation(text: str)) Extract potential abbreviation from text
- (create_entity_pipeline()) Factory function to create pipeline
- (quick_resolve_customer()) Quick customer name resolution

pdf_parser.py

```
pdf_parser.py

—— PDFProcessor

—— Library Support Detection

—— Content Extraction

—— Capability Management
```

PDF Parser Module: Handles PDF detection, downloading, and text extraction with multiple library support.

Class: PDFProcessor

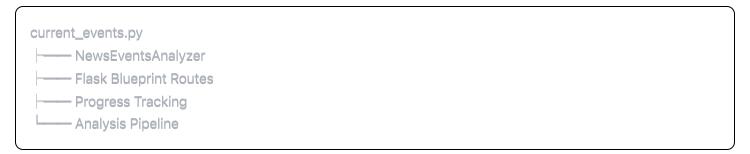
- __init__(max_pages: int, max_content_length: int)
- (_check_capabilities()) Check available PDF parsing libraries
- (get_capabilities()) Get processor capabilities and recommendations
- (is_pdf_url(url: str)) Check if URL points to PDF
- (process_pdf_from_url(url: str)) Download and process PDF from URL
- (process_pdf_file(file_path: str)) Process local PDF file
- _extract_with_pypdf2()) Extract text using PyPDF2
- _extract_with_pdfplumber()) Extract text using pdfplumber
- _extract_with_pymupdf() Extract text using PyMuPDF
- (_clean_extracted_text()) Clean and normalize extracted text

Functions:

- (is_pdf_url(url: str)) Global function to check PDF URLs
- (process_pdf(source: str)) Global function to process PDF from URL or file

• (get_pdf_capabilities()) - Get global PDF processing capabilities

current_events.py



Current Events Analysis: Analyzes current events and news with Al-powered insights and trend detection.

Class: NewsEventsAnalyzer

- (__init__(data_dir: str, topics_dir: str))
- (start_analysis(session_id: str, chain_context: bool)) Start current events analysis
- (get_progress(session_id: str)) Get analysis progress
- (_run_complete_analysis()) Run complete analysis pipeline
- _collect_recent_content()) Collect recent content from sources
- (_analyze_with_ai()) Analyze content with Al
- (_save_analysis_results()) Save analysis results to file

Flask Routes:

- (start_current_events_analysis()) Start analysis endpoint
- (get_current_events_progress()) Get progress endpoint
- (get_current_events_results()) Get results endpoint
- (health_check()) Health check endpoint

Functions:

- (set_data_directory()) Set data directory for module
- (set_ollama_model()) Set Ollama model for analysis

enhanced_chat_manager.py

enhanced_chat_manager.py

----- EnhancedChatManager

----- Message Processing

----- Context Management

----- Customer Correlation

Enhanced Chat Management: Provides advanced chat capabilities with customer correlation and context awareness.

Class: EnhancedChatManager

- (__init__(content_curator, chat_assistant, data_directory: str))
- (process_enhanced_message(message: str, session_id: str)) Process message with enhancements
- (_detect_customer_correlation_request(message: str)) Detect customer correlation requests
- _correlate_with_customers(message: str, session_id: str)) Correlate content with customers
- (_is_customer_name_in_message(message: str)) Check for customer names in message
- _enhance_basic_response() Enhance basic chat responses with context

Factory Functions:

- Create_enhanced_chat_manager()) Create enhanced chat manager instance
- (add_enhanced_chat_endpoints()) Add enhanced chat endpoints to Flask app

chain_operations_manager.py

chain_operations_manager.py

—— ChainOperationsManager

—— Operation Orchestration

—— Progress Management

L—— Flask Integration

Chain Operations Manager: Orchestrates complex multi-step operations like "What's Going On" and "Week Review".

Class: ChainOperationsManager

- __init__(content_curator, current_events_module, week_review_processor, podcast_generator)
- (start_whats_going_on(session_id: str, force_refresh: bool) Start "What's Going On" operation
- (start_week_review(session_id: str)) Start week review operation

- (start_podcast_generation(session_id: str)) Start podcast generation
- (get_progress(session_id: str)) Get operation progress
- (_orchestrate_whats_going_on()) Orchestrate "What's Going On" workflow
- (_check_existing_content()) Check for existing content
- (_format_completion_message()) Format completion messages

Factory Functions:

- (create_chain_operations_manager()) Create chain operations manager
- (add_chain_operations_endpoints()) Add Flask endpoints for chain operations

week_review_integration.py

Week Review Integration: Processes weekly analysis by aggregating and analyzing multiple days of current events data.

Class: WeekInReviewProcessor

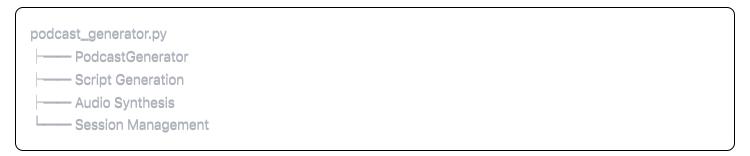
- (__init__(chat_assistant, research_assistant, data_directory: str))
- (start_week_review(session_id: str, chain_context: bool)) Start week review analysis
- (get_progress(session_id: str)) Get processing progress
- (_run_week_review_analysis()) Run complete analysis pipeline
- (_find_recent_analysis_files()) Find recent analysis files
- (_parse_analysis_file()) Parse individual analysis file
- (_aggregate_daily_data()) Aggregate data from multiple days
- _generate_week_summary()) Generate comprehensive week summary

Functions:

- (initialize_week_review()) Initialize week review processor
- (add_week_review_endpoints()) Add Flask endpoints

• (create_week_review_processor()) - Factory function for processor creation

podcast_generator.py



Podcast Generator: Creates daily podcasts from current events analysis with Al-generated scripts and audio synthesis.

Class: PodcastSession

- (__init__(session_id: str, topics_directory: str, podcasts_directory: str))
- (to_dict()) Convert session to dictionary representation

Class: PodcastGenerator

- (__init__(topics_directory: str, podcasts_directory: str, chat_assistant))
- (start_podcast_generation(session_id: str, chain_context: bool)) Start podcast generation
- (get_progress(session_id: str)) Get generation progress
- _generate_podcast() Generate complete podcast
- (_load_current_events_data()) Load current events analysis data
- _generate_script()) Generate podcast script with AI
- (_synthesize_audio()) Synthesize audio from script
- (_save_metadata()) Save podcast metadata

Functions:

- (create_podcast_generator()) Factory function to create generator
- (add_podcast_endpoints()) Add Flask endpoints for podcast operations

data_access_helper.py

Data Access Helper: Provides cross-referencing capabilities and historical data access across the system.

Class: ArticleCrossReferencer

- (__init__(data_directory: str))
- (cross_reference_articles(query: str, max_results: int)) Cross-reference articles
- (find_related_content(article_url: str)) Find related content
- (get_article_context(article_id: str)) Get article context and metadata

Class: HistoricalDataAccessor

- (__init__(data_directory: str, topics_directory: str, podcasts_directory: str))
- (get_data_summary(start_date: datetime, end_date: datetime)) Get data summary for date range
- (find_files_by_pattern(pattern: str, directory: str)) Find files matching pattern
- (get_recent_analysis(days_back: int)) Get recent analysis files
- _date_range(start_date: datetime, end_date: datetime)) Generate date range

Functions:

(create_data_access_components()) - Create data access helper components

Summary

This codebase implements a sophisticated content curation and analysis system with the following key architectural components:

- 1. **Two-Track Architecture**: Separates Chain Operations (complex workflows) from Enhanced Chat (interactive messaging)
- 2. Content Curation: Automated collection from RSS feeds, arXiv, and web sources
- 3. **Customer Analysis**: Advanced customer insights with battle card generation
- 4. Entity Resolution: Fuzzy matching and disambiguation of customer names and intents

- 5. Multi-format Support: PDF, Word, Excel, PowerPoint, HTML, and Markdown processing
- 6. Al Integration: Ollama-based Al for content analysis, summarization, and insights
- 7. **Session Management**: Persistent sessions with context awareness
- 8. **Progress Tracking**: Real-time progress updates for long-running operations
- 9. Flask Web Interface: RESTful API with comprehensive endpoint coverage

The system is designed for scalability, maintainability, and extensibility with clear separation of concerns and modular architecture.