

Documentation

Table of Contents

[local-deepwiki-mcp](#)

[Architecture Documentation](#)

[Dependencies Overview](#)

[Modules](#)

[src Module Documentation](#)

[Tests Module](#)

[Source Files](#)

[config.py](#)

[server.py](#)

[watcher.py](#)

[Parser Module](#)

[vectorstore.py](#)

[PDF Export Module](#)

[API Documentation Generator](#)

[CrossLinks Module](#)

[diagrams.py](#)

[Manifest Generator Module](#)

[Wiki Generator Module](#)

[test_api_docs.py](#)

[test_crosslinks.py](#)

[test_indexer.py](#)

[test_manifest.py](#)

[test_models.py](#)

[test_parser.py](#)

[test_pdf_export.py](#)

[test_retry.py](#)

[test_vectorstore.py](#)

[File Overview](#)

[Indexer Module Documentation](#)

[File Overview](#)

[Search Index Generator](#)

[File Overview](#)

[File Overview](#)

[Base Provider Classes](#)

[Local Embedding Provider](#)

[OpenAI Embedding Provider](#)

[Anthropic Provider Documentation](#)

[Ollama Provider Documentation](#)

[OpenAI LLM Provider](#)

[File Overview](#)

[File Overview](#)

[File Overview](#)

[Test Configuration File Documentation](#)

[File Overview](#)

[File Overview](#)

[File Overview](#)

[File: test_search.py](#)

[File Overview](#)

[File Overview](#)

[File Overview](#)

[Test Web Documentation](#)

local-deepwiki-mcp

Local DeepWiki-style MCP server for private repository documentation

Description

Local DeepWiki MCP is a Model Context Protocol (MCP) server that generates documentation for private repositories. It provides both a server interface for MCP clients and a web application for browsing generated documentation, with support for multiple LLM providers including Ollama, OpenAI, and Anthropic.

Key Features

- **MCP Server Integration** - Implements a Model Context Protocol server with stdio communication for integration with MCP clients
- **Multiple LLM Provider Support** - Configurable support for Ollama (with local hosting), OpenAI, and Anthropic language models through dedicated provider classes
- **Repository Indexing** - Core indexer functionality that processes repository contents with configurable embedding providers (local or OpenAI)
- **Web Interface** - Built-in Flask web server for browsing and serving generated wiki documentation
- **Export Capabilities** - Multiple export formats including HTML and PDF through dedicated command-line tools

Technology Stack

- **Python >=3.11**
- **Dependencies:** anthropic, flask, lancedb, markdown, mcp, ollama, openai, pandas, pydantic, pyyaml, rich, sentence-transformers
- Plus 16 more...

Directory Structure

```
local-deepwiki-mcp/
├── html-export/
│   ├── files/
│   ├── modules/
│   ├── architecture.html
│   ├── dependencies.html
│   ├── index.html
│   └── search.json
├── src/
│   └── local_deepwiki/
└── tests/
    ├── __init__.py
    ├── test_api_docs.py
    ├── test_callgraph.py
    ├── test_chunker.py
    ├── test_config.py
    ├── test_crosslinks.py
    ├── test_diagrams.py
    ├── test_html_export.py
    ├── test_incremental_wiki.py
    ├── test_indexer.py
    ├── test_manifest.py
    ├── test_models.py
    ├── test_ollama_health.py
    └── test_parser.py
...
...
```

Quick Start

- `deepwiki-export` → `local_deepwiki.export.html:main`
- `deepwiki-export-pdf` → `local_deepwiki.export.pdf:main`
- `deepwiki-serve` → `local_deepwiki.web.app:main`
- `deepwiki-watch` → `local_deepwiki.watcher:main`
- `local-deepwiki` → `local_deepwiki.server:main`

Architecture Documentation

System Overview

Local DeepWiki is a documentation generation system that processes codebases to create comprehensive wiki-style documentation. The system is built around a modular architecture with configurable providers for language models and embeddings, supporting multiple AI providers including Ollama, Anthropic, and OpenAI.

The core workflow involves parsing code into chunks, generating embeddings for semantic search, and using large language models to create structured documentation with architecture diagrams and cross-references.

Key Components

Configuration Management

The [Config](#) class serves as the central configuration hub, managing settings for all system components. It coordinates with specialized configuration classes:

- [LLMConfig](#) manages language model provider selection and settings
- [EmbeddingConfig](#) handles embedding provider configuration
- [OllamaConfig](#), [AnthropicConfig](#), and [OpenAILLMConfig](#) provide provider-specific settings

The configuration system supports context-aware overrides through the [`config_context`](#) function and maintains thread-safe global state.

Code Processing

[CodeChunker](#) breaks down source code into semantic units represented by [ChunkType](#) enumeration values (function, class, method, module, import, comment, other). This chunking enables fine-grained analysis and documentation generation.

[ProjectManifest](#) analyzes project structure and dependencies, providing technology stack summaries and entry point identification for architectural documentation.

LLM Provider System

The system implements a provider pattern for language model integration:

- **LLMProvider** defines the abstract interface for language model operations
- **OllamaProvider** implements local Ollama model integration with health checking
- The `get_llm_provider` factory function creates appropriate provider instances based on configuration

Documentation Generation

WikiGenerator orchestrates the documentation creation process, including architecture documentation generation through the `_generate_architecture` method. It coordinates with vector stores for semantic search and uses configured LLM providers to generate comprehensive documentation.

EntityRegistry manages code entities and their relationships for cross-referencing and navigation.

Utility Components

DebouncedHandler provides rate-limiting capabilities for operations that may be triggered frequently.

The **with_retry** decorator implements retry logic for operations that may fail transiently, particularly useful for network-based provider operations.

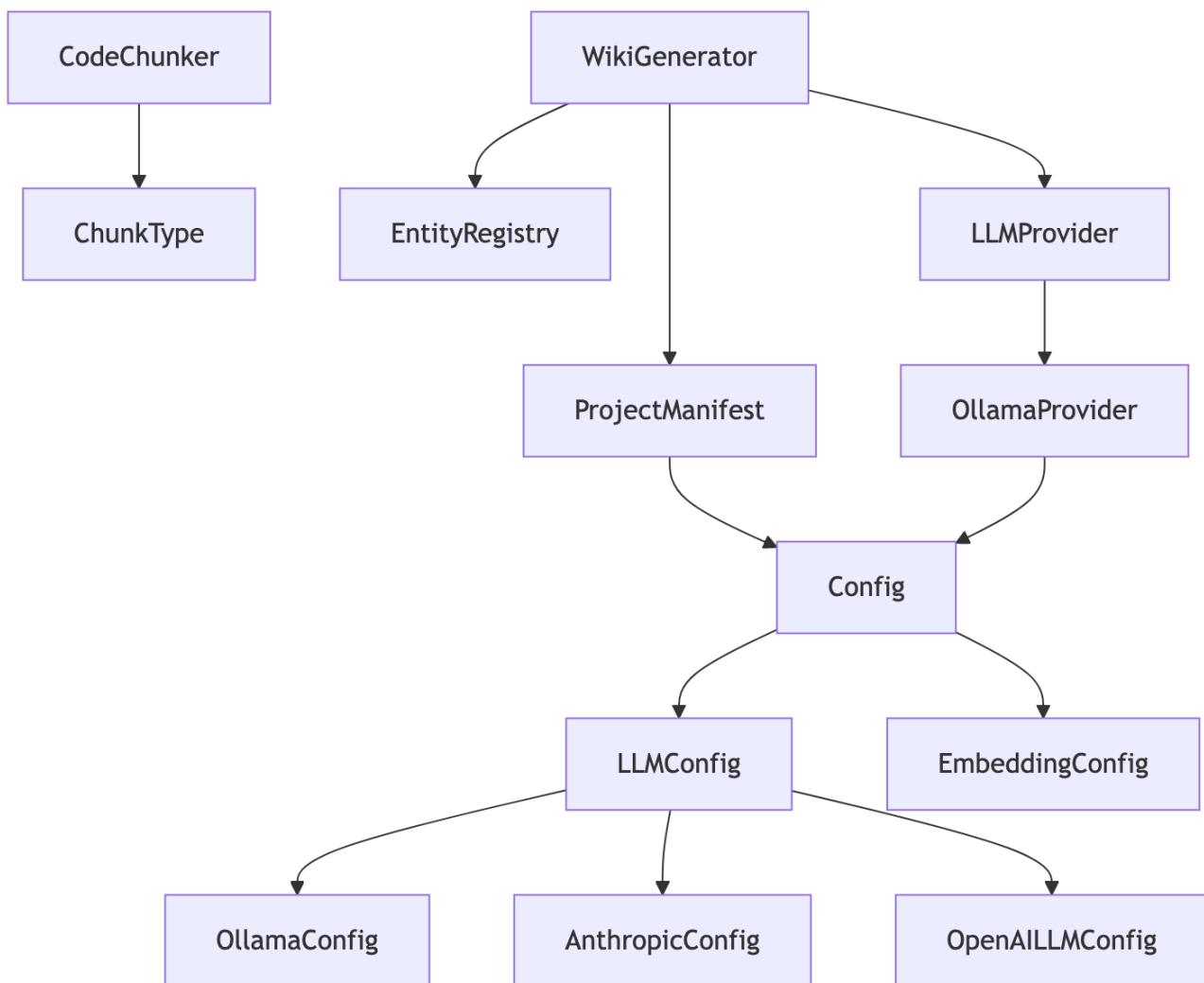
Data Flow

1. **Configuration Loading:** The system loads configuration through **Config** and its specialized sub-configurations
2. **Code Parsing:** CodeChunker processes source files into semantic chunks categorized by ChunkType
3. **Project Analysis:** **ProjectManifest** analyzes the codebase structure and dependencies
4. **Entity Registration:** **EntityRegistry** catalogs discovered code entities and their relationships
5. **Vector Storage:** Code chunks are embedded and stored for semantic search capabilities

6. **Documentation Generation:** `WikiGenerator` uses LLM providers to create documentation, leveraging vector search for context gathering

7. **Architecture Analysis:** The `_generate_architecture` method performs multiple semantic searches to gather comprehensive architectural context

Component Diagram



Key Design Decisions

Provider Pattern Implementation

The system uses a provider pattern for LLM integration, allowing seamless switching between different AI services (Ollama, Anthropic, OpenAI) through configuration. The `get_llm_provider` factory function centralizes provider instantiation logic.

Configuration Context Management

The configuration system supports both global and context-specific settings through the `config_context` context manager, enabling different configurations for different operations without affecting global state.

Semantic Code Chunking

Code is processed into semantic chunks categorized by `ChunkType` enumeration, enabling more intelligent documentation generation by preserving code structure and meaning rather than arbitrary text splitting.

Health Checking for External Services

The `OllamaProvider` implements health checking through the `check_health` and `_ensure_healthy` methods, ensuring robust operation when working with external services that may be unavailable.

Multi-Modal Search Strategy

The `_generate_architecture` method demonstrates a sophisticated approach to gathering architectural context by performing multiple targeted searches (core components, architectural patterns, data flow) rather than a single broad search.

Retry Mechanism

The `with_retry` decorator provides configurable retry logic for operations that may fail due to network issues or temporary service unavailability, improving system reliability.

Relevant Source Files

The following source files were used to generate this documentation:

- [tests/test_parser.py:24–123](#)
- [tests/test_retry.py:8–144](#)
- [tests/test_ollama_health.py:13–32](#)
- [tests/test_server_handlers.py:15–69](#)
- [tests/test_chunker.py:11–182](#)
- [tests/test_vectorstore.py:9–28](#)
- [tests/test_pdf_export.py:21–80](#)
- [tests/test_search.py:20–53](#)
- [tests/test_toc.py:17–43](#)
- [tests/test_incremental_wiki.py:20–47](#)

Showing 10 of 63 source files.

Dependencies Overview

External Dependencies

The project relies on the following third-party libraries:

AI and Language Processing

- **anthropic** (>=0.40) - Anthropic's Claude API client for AI language model interactions
- **openai** (>=1.0) - OpenAI API client for GPT models and embeddings
- **ollama** (>=0.4) - Local LLM inference server client
- **sentence-transformers** (>=3.0) - Pre-trained models for generating text embeddings
- **mcp** (>=1.2.0) - Model Context Protocol for AI model communication

Code Analysis and Parsing

- **tree-sitter** (>=0.23) - Incremental parsing library for syntax trees
- **tree-sitter-c** (>=0.23) - C language grammar for tree-sitter
- **tree-sitter-c-sharp** (>=0.23) - C# language grammar
- **tree-sitter-cpp** (>=0.23) - C++ language grammar
- **tree-sitter-go** (>=0.23) - Go language grammar
- **tree-sitter-java** (>=0.23) - Java language grammar
- **tree-sitter-javascript** (>=0.23) - JavaScript language grammar
- **tree-sitter-kotlin** (>=0.23) - Kotlin language grammar
- **tree-sitter-php** (>=0.23) - PHP language grammar
- **tree-sitter-python** (>=0.23) - Python language grammar
- **tree-sitter-ruby** (>=0.23) - Ruby language grammar
- **tree-sitter-rust** (>=0.23) - Rust language grammar
- **tree-sitter-swift** (>=0.0.1) - Swift language grammar
- **tree-sitter-typescript** (>=0.23) - TypeScript language grammar

Data Storage and Processing

- **lancedb** (>=0.15) - Vector database for embeddings storage and similarity search
- **pandas** (>=2.0) - Data manipulation and analysis library

Web and Document Generation

- **flask** (>=3.0) - Web framework for server functionality
- **markdown** (>=3.0) - Markdown to HTML conversion
- **weasyprint** (>=62.0) - HTML/CSS to PDF rendering

Utilities

- **pydantic** (>=2.0) - Data validation and settings management
- **pyyaml** (>=6.0) - YAML file parsing and generation
- **rich** (>=13.0) - Rich text and beautiful formatting in terminal
- **watchdog** (>=4.0) - File system event monitoring

Dev Dependencies

Development and testing tools:

- **black** (>=24.0) - Python code formatter
- **isort** (>=5.0) - Import statement organizer
- **mypy** (>=1.0) - Static type checker
- **pre-commit** (>=3.0) - Git pre-commit hooks framework
- **pytest** (>=8.0) - Testing framework
- **pytest-asyncio** (>=0.24) - Async testing support for pytest

Internal Module Dependencies

Based on the import statements, the internal module structure shows these key relationships:

Core Modules

- **CodeChunker** depends on [CodeParser](#) for parsing functionality and uses models like ChunkType, CodeChunk, and Language
- [CodeParser](#) provides parsing utilities used by multiple generators and the chunker
- [VectorStore](#) works with embedding providers and code chunks for similarity search
- [RepositoryIndexer](#) coordinates chunking, parsing, and indexing operations

Generator Modules

- [CrossLinker](#) and [EntityRegistry](#) work with WikiPage and CodeChunk models to create cross-references
- [APIDocExtractor](#) uses [CodeParser](#) to extract API documentation from code
- [RelationshipAnalyzer](#) and FileRelationships analyze code relationships for "See Also" sections
- Multiple generators (diagrams, search, source refs, TOC) depend on core models and parsing functionality

Provider System

- [EmbeddingProvider](#) and [LLMProvider](#) serve as base classes for AI service integrations
- [LocalEmbeddingProvider](#) uses sentence-transformers for local embeddings
- [OpenAIEmbeddingProvider](#) integrates with OpenAI's embedding API

Server Components

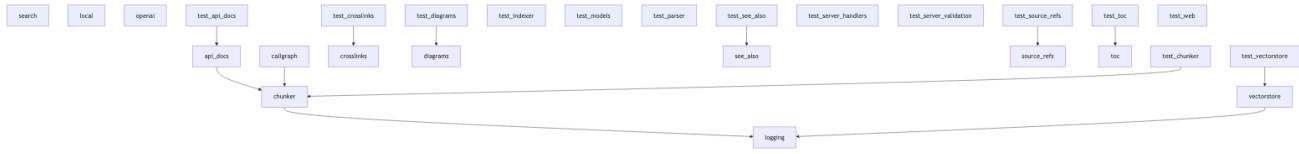
- Server handlers coordinate various generators and core functionality for web API endpoints

The architecture follows a layered approach where core parsing and chunking functionality supports multiple specialized generators, all coordinated through a provider system for AI services and a web server for user interaction.

Module Dependency Graph

The following diagram shows internal module dependencies:

Dependencies Overview



Relevant Source Files

The following source files were used to generate this documentation:

- [tests/test_parser.py:24–123](#)
- [tests/test_retry.py:8–144](#)
- [tests/test_ollama_health.py:13–32](#)
- [tests/test_server_handlers.py:15–69](#)
- [tests/test_chunker.py:11–182](#)
- [tests/test_vectorstore.py:9–28](#)
- [tests/test_pdf_export.py:21–80](#)
- [tests/test_search.py:20–53](#)
- [tests/test_toc.py:17–43](#)
- [tests/test_incremental_wiki.py:20–47](#)

Showing 10 of 63 source files.

Modules

This section contains documentation for each module.

- [Module: tests](#)
- [Module: src](#)

src Module Documentation

Module Purpose

The `src` module contains the core implementation of the local_deepwiki system, a documentation generation and management tool for codebases. Based on the code structure, it provides functionality for parsing source code, generating documentation, managing vector stores, and serving content through a web interface.

Key Classes and Functions

Core Components

CodeChunk (from core/chunker.py)

The chunker module contains methods for creating code chunks from parsed source files:

- `_create_module_chunk` - Creates a chunk representing a module or file overview from an AST root node, source bytes, language information, and file path.

Generators

ManifestCacheEntry and ProjectManifest (from generators/manifest.py)

Classes for managing project manifest information and caching:

- `ManifestCacheEntry` - Dataclass for storing cached manifest entries
- `ProjectManifest` - Handles project manifest data

The manifest module also includes utility functions: - `_get_manifest_mtimes` - Retrieves modification times for manifest files - `_is_cache_valid` - Validates cache validity - `_load_manifest_cache` - Loads cached manifest data - `save` - Saves manifest data (function name truncated in context)

Source References Generator (generators/source_refs.py)

Functions for generating source code reference sections:

- **build_file_to_wiki_map** - Creates mapping between files and wiki pages
- **_relative_path** - Converts paths to relative format
- **_format_file_entry** - Formats file entries for display
- **generate_source_refs_section** - Generates source reference sections
- **add_source_refs_sections** - Adds source reference sections to content

See Also Generator (generators/see_also.py)

Contains methods for generating "see also" sections:

- **_module_matches_file** - Checks if a module name corresponds to a specific file path by converting file paths to module-like format

Diagrams Generator (generators/diagrams.py)

Utility functions for diagram generation:

- **_path_to_module** - Converts file paths (like `src/local_deepwiki/core/indexer.py`) to module names (like `core.indexer`), filtering out non-Python files and files starting with double underscores

Providers

Base Provider Classes (providers/init.py)

The providers module exports base classes for different service providers:

- **EmbeddingProvider** - Base class for embedding service providers
- **LLMProvider** - Base class for language model providers

How Components Interact

The components work together to create a comprehensive documentation system:

1. **Code Processing:** The chunker processes source code files, creating CodeChunk objects that represent different parts of the codebase
2. **Manifest Management:** The manifest system tracks project configuration and caches metadata for efficient processing
3. **Reference Generation:** Source reference generators create cross-links between documentation and source files
4. **Provider Abstraction:** The provider system allows pluggable backends for embeddings and language models

Usage Examples

Creating Module Chunks

```
# Example of how _create_module_chunk might be used
chunk = chunker._create_module_chunk(
    root=ast_root_node,
    source=file_contents.encode(),
    language=python_language,
    file_path="src/local_deepwiki/core/indexer.py"
)
```

Converting Paths to Modules

```
# Convert a file path to module name
module_name = _path_to_module("src/local_deepwiki/core/indexer.py")
# Returns: "core.indexer"
```

Checking Module-File Matches

```
# Check if a module corresponds to a file
matches = generator._module_matches_file(
    "local_deepwiki.core.chunker",
    "src/local_deepwiki/core/chunker.py"
)
```

Building File-Wiki Mappings

```
# Generate mapping between source files and wiki pages
file_map = build_file_to_wiki_map(wiki_pages)
```

Dependencies

Based on the imports shown in the code context:

- **Standard Library:** `json`, `re`, `pathlib.Path`, `dataclasses`, `typing`
- **TOML Processing:** `tomllib` (with fallback to `tomli`)
- **Internal Dependencies:**
 - `local_deepwiki.logging` - For logging functionality
 - `local_deepwiki.models` - For `WikiPage` and `WikiPageStatus` models
 - `local_deepwiki.providers.base` - For provider base classes

The module structure suggests a well-organized codebase with clear separation between core functionality, generators for different types of content, web interface components, and pluggable provider systems.

Relevant Source Files

The following source files were used to generate this documentation:

- `src/local_deepwiki/logging.py:19-70`
- `src/local_deepwiki/server.py:33-53`
- `src/local_deepwiki/config.py:13-18`

- [src/local_deepwiki/models.py:11–26](#)
- [src/local_deepwiki/__init__.py](#)
- [src/local_deepwiki/watcher.py:29–223](#)
- [src/local_deepwiki/tools/__init__.py](#)
- [src/local_deepwiki/core/chunker.py:200–597](#)
- [src/local_deepwiki/core/vectorstore.py:37–395](#)
- [src/local_deepwiki/core/__init__.py](#)

Showing 10 of 37 source files.

Tests Module

Module Purpose

The tests module contains the test suite for the local_deepwiki project, providing comprehensive unit and integration tests for various components including parsers, generators, web interfaces, and external service integrations.

Key Classes and Functions

Parser Testing

TestCodeParser Tests the `CodeParser` functionality, including language detection capabilities. The test suite verifies that the parser can correctly identify Python files by their extensions.

TestPathToModule Tests the `_path_to_module` function which converts file paths to module names. Includes tests for basic path conversion and handling of special files like `__init__.py`.

Generator Testing

TestProjectManifest Tests manifest generation functionality for project metadata extraction.

TestAddSourceRefsSections Tests the addition of source reference sections to generated documentation pages. Verifies proper handling of different page types including file pages, index pages, and module pages.

Web Interface Testing

TestBuildBreadcrumb Tests breadcrumb navigation generation for the web interface.

TestFlaskApp Tests Flask application functionality and routing.

TestTemplateConfiguration Tests template configuration for the web interface.

Server Handler Testing

TestHandleIndexRepository Tests repository indexing functionality.

TestHandleAskQuestion Tests question handling capabilities.

TestHandleSearchCode Tests code search functionality.

TestHandleReadWikiStructure Tests wiki structure reading operations.

TestHandleReadWikiPage Tests individual wiki page reading operations.

TestHandleExportWikiHtml Tests HTML export functionality.

External Service Testing

TestOllamaConnectionError Tests error handling for Ollama connection issues.

TestOllamaModelErrorNotFoundError Tests error handling when Ollama models are not found.

TestOllamaProviderHealthCheck Tests health checking for the Ollama service provider.

TestOllamaProviderGenerate Tests text generation via the Ollama provider.

TestOllamaProviderGenerateStream Tests streaming text generation via the Ollama provider.

Cross-linking and Analysis Testing

TestEntityRegistry Tests entity registration and lookup functionality for cross-linking between documentation pages. Includes tests for entity registration, name filtering, and alias lookup.

TestExtractCallsPython Tests call graph extraction from Python code, including function calls, method calls, and nested call detection.

How Components Interact

The test classes work together to provide comprehensive coverage of the application's functionality:

1. **Parser tests** verify that code analysis components correctly process source files
2. **Generator tests** ensure that documentation generation works properly
3. **Web interface tests** validate the user-facing components

4. **Server handler tests** verify API endpoints and request processing
5. **External service tests** ensure proper integration with services like Ollama

Usage Examples

Running Parser Tests

```
# Test language detection
parser = CodeParser()
result = parser.detect_language(Path("test.py"))
assert result == Language.PYTHON
```

Testing WikiPage Model

```
# Test WikiPage representation
page = WikiPage(
    path="modules/core.md",
    title="Core Module",
    content="# Core Module\n\nContent here.",
    generated_at=1234567890.0,
)
result = repr(page)
assert "WikiPage" in result
```

Testing Web Interface

```
# Test breadcrumb generation
breadcrumb = buildBreadcrumb("/modules/core/parser")
# Verify breadcrumb structure
```

Dependencies

Based on the imports shown, the tests module depends on:

- **Standard library:** json, tempfile, time, pathlib.Path
- **Testing framework:** pytest

- **Mocking:** `unittest.mock` (`AsyncMock`, `patch`, `MagicMock`)

- **Application modules:**

- `local_deepwiki.generators.manifest`
- `local_deepwiki.providers.llm.ollama`
- `local_deepwiki.web.app`
- `local_deepwiki.server`

The test suite uses `pytest` as the primary testing framework with extensive use of mocking for external dependencies and asynchronous operations.

Relevant Source Files

The following source files were used to generate this documentation:

- [tests/test_parser.py:24–123](#)
- [tests/test_retry.py:8–144](#)
- [tests/test_ollama_health.py:13–32](#)
- [tests/test_server_handlers.py:15–69](#)
- [tests/test_chunker.py:11–182](#)
- [tests/test_vectorstore.py:9–28](#)
- [tests/test_pdf_export.py:21–80](#)
- [tests/test_search.py:20–53](#)
- [tests/test_toc.py:17–43](#)
- [tests/test_incremental_wiki.py:20–47](#)

Showing 10 of 26 source files.

Source Files

Detailed documentation for individual source files.

src

- [api_docs.py](#)
- [config.py](#)
- [crosslinks.py](#)
- [diagrams.py](#)
- [manifest.py](#)
- [parser.py](#)
- [pdf.py](#)
- [server.py](#)
- [vectorstore.py](#)
- [watcher.py](#)
- [wiki.py](#)

tests

- [test_api_docs.py](#)
- [test_crosslinks.py](#)
- [test_indexer.py](#)
- [test_manifest.py](#)
- [test_models.py](#)
- [test_parser.py](#)
- [test_pdf_export.py](#)
- [test_retry.py](#)
- [test_vectorstore.py](#)

config.py

File Overview

This module provides configuration management for the local_deepwiki application using Pydantic models and context variables. It handles various configuration aspects including embedding providers, language models, parsing settings, chunking parameters, and output configurations. The module implements thread-safe configuration management with context-aware configuration switching.

Classes

LocalEmbeddingConfig

A Pydantic model for configuring local embedding settings. This class handles configuration for locally-hosted embedding models.

OpenAIEmbeddingConfig

A Pydantic model for configuring OpenAI embedding API settings. This class manages OpenAI-specific embedding parameters and authentication.

EmbeddingConfig

A unified configuration model that handles different embedding provider configurations. This class serves as a wrapper for various embedding backends.

OllamaConfig

A Pydantic model for configuring Ollama language model settings. This class manages connection and model parameters for Ollama instances.

AnthropicConfig

A Pydantic model for configuring Anthropic language model settings. This class handles Anthropic API configuration and model parameters.

OpenAILLMConfig

A Pydantic model for configuring OpenAI language model settings. This class manages OpenAI LLM API parameters and authentication.

LLMConfig

A unified configuration model that handles different language model provider configurations. This class serves as a wrapper for various LLM backends.

ParsingConfig

A Pydantic model for configuring document parsing settings. This class manages parameters related to how documents are processed and parsed.

ChunkingConfig

A Pydantic model for configuring text chunking parameters. This class handles settings for how text is split into chunks for processing.

WikiConfig

A Pydantic model for configuring wiki-specific settings. This class manages parameters related to wiki generation and formatting.

OutputConfig

A Pydantic model for configuring output settings. This class handles parameters for how results are formatted and saved.

Config

The `main` configuration class that combines all configuration sections. This class serves as the root configuration model containing all other configuration components.

Functions

`get_config()`

Retrieves the current configuration from the context variable.

Returns: The current Config instance

`set_config(config)`

Sets the configuration in the current context.

Parameters: - `config` : The Config instance to set as current

`reset_config()`

Resets the configuration context variable to its default state.

`config_context(config)`

A context manager that temporarily sets a configuration for the duration of the context.

Parameters: - `config` : The Config instance to use within the context

Returns: A context manager that handles configuration switching

Usage Examples

Basic Configuration Access

```
from local_deepwiki.config import get_config, set_config

# Get current configuration
current_config = get_config()

# Set a new configuration
new_config = Config(...)
set_config(new_config)
```

Using Configuration Context

```
from local_deepwiki.config import config_context, Config

# Temporarily use a different configuration
with config_context(temp_config):
    # Operations here use temp_config
    current = get_config() # Returns temp_config

# Outside context, original config is restored
```

Configuration Reset

```
from local_deepwiki.config import reset_config

# Reset to default configuration
reset_config()
```

Related Components

This module integrates with several external libraries:

- **pydantic**: Used for configuration model validation and serialization through BaseModel and Field
- **yaml**: Handles YAML configuration file parsing
- **pathlib**: Manages file system paths through the Path class
- **threading**: Provides thread-safe configuration management
- **contextlib**: Enables context manager functionality for configuration switching
- **contextvars**: Implements context-aware configuration storage

The configuration models support various provider integrations including OpenAI, Anthropic, and Ollama services, making this module central to the application's provider abstraction layer.

API Reference

class LocalEmbeddingConfig

Inherits from: BaseModel

Configuration for local embedding model.

class OpenAIEEmbeddingConfig

Inherits from: BaseModel

Configuration for OpenAI embedding model.

class EmbeddingConfig

Inherits from: BaseModel

Embedding provider configuration.

class OllamaConfig

Inherits from: BaseModel

Configuration for Ollama LLM.

class AnthropicConfig**Inherits from:** BaseModel

Configuration for Anthropic LLM.

class OpenAILLMConfig**Inherits from:** BaseModel

Configuration for OpenAI LLM.

class LLMConfig**Inherits from:** BaseModel

LLM provider configuration.

class ParsingConfig**Inherits from:** BaseModel

Code parsing configuration.

class ChunkingConfig**Inherits from:** BaseModel

Chunking configuration.

class WikiConfig**Inherits from:** BaseModel

Wiki generation configuration.

class OutputConfig**Inherits from:** BaseModel

Output configuration.

class Config**Inherits from:** BaseModel

Main configuration.

Methods:**load**

```
def load(config_path: Path | None = None) -> "Config"
```

Load configuration from file or defaults.

Parameter	Type	Default	Description
config_path	Path None	None	-

get_wiki_path

```
def get_wiki_path(repo_path: Path) -> Path
```

Get the wiki output path for a repository.

Parameter	Type	Default	Description
repo_path	Path	-	-

get_vector_db_path

```
def get_vector_db_path(repo_path: Path) -> Path
```

Get the vector database path for a repository.

Parameter	Type	Default	Description
repo_path	Path	-	-

Functions

`get_config`

```
def get_config() -> Config
```

Get the configuration instance. Returns the context-local config if set, otherwise the global config. Thread-safe for concurrent access.

Returns: Config

`set_config`

```
def set_config(config: Config) -> None
```

Set the global configuration instance. Thread-safe. Note: This sets the global config, not a context-local one. Use `config_context()` for temporary context-local overrides.

Parameter	Type	Default	Description
<code>config</code>	Config	-	The configuration to set globally.

Returns: None

`reset_config`

```
def reset_config() -> None
```

Reset the global configuration to uninitialized state. Useful for testing to ensure a fresh config is loaded. Also clears any context-local override.

Returns: None

`config_context`

@contextmanager

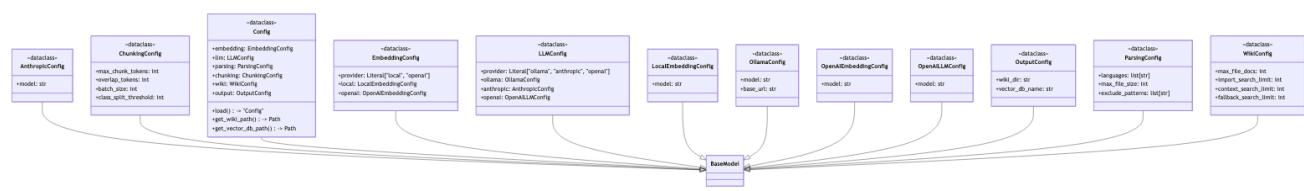
```
def config_context(config: Config) -> Generator[Config, None, None]
```

Context manager for temporary config override. Sets a context-local configuration that takes precedence over the global config within the context. Useful for testing or per-request config.

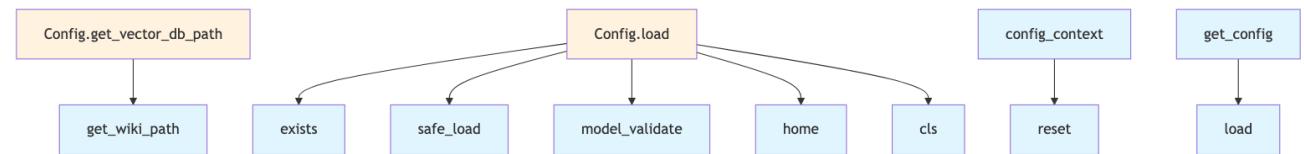
Parameter	Type	Default	Description
config	Config	-	The configuration to use within the context.

Returns: Generator[Config, None, None]

Class Diagram



Call Graph



Relevant Source Files

- src/local_deepwiki/config.py:13-18

See Also

- [server](#) - uses this
- [test_indexer](#) - uses this
- [diagrams](#) - shares 2 dependencies

server.py

File Overview

The server.py file implements an MCP (Model Context Protocol) server that provides tools for repository indexing and wiki generation. It serves as the main server interface for the local-deepwiki system, exposing functionality through MCP tools that can be called by compatible clients.

Functions

Validation Functions

`_validate_positive_int`

Validates that a value is a positive integer.

`_validate_non_empty_string`

Validates that a value is a non-empty string.

`_validate_language`

Validates that a language value is a valid Language enum member.

`_validate_languages_list`

Validates that a value is a list of valid Language enum members.

`_validate_provider`

Validates that a provider value is supported.

MCP Server Functions

`list_tools`

Returns the list of available MCP tools that the server provides. This function defines the tools interface for the MCP server.

`call_tool`

Handles incoming tool calls from MCP clients. Routes tool requests to the appropriate handler functions based on the tool name.

Tool Handlers

`handle_index_repository`

Handles the repository indexing tool call. This function:

- Takes repository path and configuration parameters
- Creates a `RepositoryIndexer` instance
- Indexes the specified repository
- Returns status information about the indexing operation

`progress_callback`

Callback function used during repository indexing to handle progress updates from the indexing process.

Related Components

The server integrates with several other components from the local-deepwiki system:

- **Config**: Uses `get_config` and `set_config` for configuration management
- **RepositoryIndexer**: Core component for indexing repository contents
- **VectorStore**: Storage system for embeddings and indexed content
- **WikiStructure and Language**: Data models for wiki generation
- **Embedding providers**: Through `get_embedding_provider` for vector embeddings
- **Wiki generation**: Through `generate_wiki` function
- **Logging**: Uses `get_logger` for logging functionality

Usage Context

This server is designed to be run as an MCP server, communicating over stdio with MCP-compatible clients. The server exposes repository indexing and wiki generation capabilities as tools that can be invoked remotely through the MCP protocol.

The server handles:

- Repository indexing with configurable parameters
- Progress tracking during long-running operations
- Error handling and validation of input parameters
- Integration with the broader local-deepwiki ecosystem

API Reference

Functions

`list_tools`

```
@server.list_tools()
```

```
async def list_tools() -> list[Tool]
```

List available tools.

Returns: `list[Tool]`

`call_tool`

```
@server.call_tool()
```

```
async def call_tool(name: str, arguments: dict[str, Any]) -> list[TextContent]
```

Handle tool calls.

Parameter	Type	Default	Description
<code>name</code>	<code>str</code>	-	-
<code>arguments</code>	<code>dict[str, Any]</code>	-	-

Returns: list[TextContent]**handle_index_repository**

```
async def handle_index_repository(args: dict[str, Any]) -> list[TextContent]
```

Handle index_repository tool call.

Parameter	Type	Default	Description
args	dict[str, Any]	-	-

Returns: list[TextContent]**progress_callback**

```
def progress_callback(msg: str, current: int, total: int)
```

Parameter	Type	Default	Description
msg	str	-	-
current	int	-	-
total	int	-	-

handle_ask_question

```
async def handle_ask_question(args: dict[str, Any]) -> list[TextContent]
```

Handle ask_question tool call.

Parameter	Type	Default	Description
args	dict[str, Any]	-	-

Returns: list[TextContent]

handle_read_wiki_structure

```
async def handle_read_wiki_structure(args: dict[str, Any]) -> list[TextContent]
```

Handle read_wiki_structure tool call.

Parameter	Type	Default	Description
args	dict[str, Any]	-	-

Returns: list[TextContent]

handle_read_wiki_page

```
async def handle_read_wiki_page(args: dict[str, Any]) -> list[TextContent]
```

Handle read_wiki_page tool call.

Parameter	Type	Default	Description
args	dict[str, Any]	-	-

Returns: list[TextContent]

handle_search_code

```
async def handle_search_code(args: dict[str, Any]) -> list[TextContent]
```

Handle search_code tool call.

Parameter	Type	Default	Description
args	dict[str, Any]	-	-

Returns: list[TextContent]

handle_export_wiki_html

```
async def handle_export_wiki_html(args: dict[str, Any]) -> list[TextContent]
```

Handle export_wiki_html tool call.

Parameter	Type	Default	Description
args	dict[str, Any]	-	-

Returns: list[TextContent]

handle_export_wiki_pdf

```
async def handle_export_wiki_pdf(args: dict[str, Any]) -> list[TextContent]
```

Handle export_wiki_pdf tool call.

Parameter	Type	Default	Description
args	dict[str, Any]	-	-

Returns: list[TextContent]

main

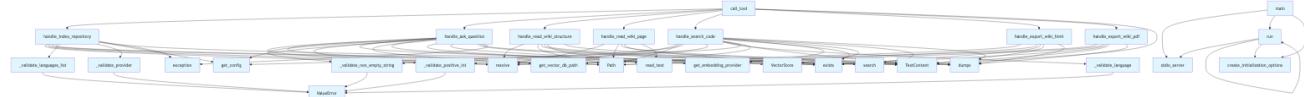
```
def main()
```

Main entry point for the MCP server.

run

```
async def run()
```

Call Graph



Relevant Source Files

- `src/local_deepwiki/server.py:33–53`

See Also

- [vectorstore](#) - dependency
- [config](#) - dependency
- [pdf](#) - dependency

watcher.py

File Overview

This file implements a file system watcher that monitors repository changes and automatically regenerates wiki documentation. It uses the watchdog library to detect file system events and triggers reindexing when relevant files are modified.

Functions

main()

Main entry point for the watch command that sets up command-line argument parsing for the file watcher.

Parameters: - None

Returns: - None

Command-line Arguments: - `repo_path` (optional): Path to the repository to watch (defaults to current directory) - `--debounce` : Seconds to wait after changes before reindexing (default: 2.0)

Usage Examples

The main function can be called directly to start the file watcher:

```
from local_deepwiki.watcher import main  
  
# Start the watcher  
main()
```

Related Components

This module integrates with several other components based on the imports:

- **Config**: Uses `get_config` function and `Config` class for configuration management
- **RepositoryIndexer**: Core indexing functionality from the indexer module
- **EXTENSION_MAP**: File extension mapping from the parser module
- **generate_wiki**: Wiki generation functionality from the generators module
- **get_logger**: Logging utilities from the logging module

The module also uses external libraries: - `watchdog` for file system monitoring - `rich.console` for console output - `argparse` for command-line interface - `asyncio` for asynchronous operations - `pathlib` for path handling

API Reference

class `DebouncedHandler`

Inherits from: `FileSystemEventHandler`

File system event handler with debouncing.

Methods:

`__init__`

```
def __init__(repo_path: Path, config: Config, debounce_seconds: float = 2.0, llm_provider: str | None = None)
```

Initialize the handler.

Parameter	Type	Default	Description
repo_path	Path	-	Path to the repository root.
config	Config	-	Configuration instance.
debounce_seconds	float	2.0	Seconds to wait after last change before triggering.
llm_provider	str None	None	Optional LLM provider override.

progress_callback

```
def progress_callback(msg: str, current: int, total: int) -> None
```

Parameter	Type	Default	Description
msg	str	-	-
current	int	-	-
total	int	-	-

on_modified

```
def on_modified(event: FileSystemEvent) -> None
```

Handle file modification events.

Parameter	Type	Default	Description
event	FileSystemEvent	-	-

on_created

```
def on_created(event: FileSystemEvent) -> None
```

Handle file creation events.

Parameter	Type	Default	Description
event	FileSystemEvent	-	-

on_deleted

```
def on_deleted(event: FileSystemEvent) -> None
```

Handle file deletion events.

Parameter	Type	Default	Description
event	FileSystemEvent	-	-

on_moved

```
def on_moved(event: FileSystemEvent) -> None
```

Handle file move events.

Parameter	Type	Default	Description
event	FileSystemEvent	-	-

class RepositoryWatcher

Watches a repository for file changes and triggers reindexing.

Methods:**__init__**

```
def __init__(repo_path: Path, config: Config | None = None, debounce_seconds: float = 2.0, llm_provi
```

Initialize the watcher.

Parameter	Type	Default	Description
repo_path	Path	-	Path to the repository to watch.
config	Config None	None	Optional configuration.
debounce_seconds	float	2.0	Seconds to wait after changes before reindexing.
llm_provider	str None	None	Optional LLM provider override.

start

```
def start() -> None
```

Start watching the repository.

stop

```
def stop() -> None
```

Stop watching the repository.

is_running

```
def is_running() -> bool
```

Check if the watcher is running.

Functions

initial_index

```
async def initial_index(repo_path: Path, config: Config, llm_provider: str | None = None, full_rebuil
```

Perform initial indexing before starting watch mode.

Parameter	Type	Default	Description
repo_path	Path	-	Path to the repository.
config	Config	-	Configuration instance.
llm_provider	str None	None	Optional LLM provider override.
full_rebuild	bool	False	Whether to do a full rebuild.

Returns: None

progress_callback

```
def progress_callback(msg: str, current: int, total: int) -> None
```

Parameter	Type	Default	Description
msg	str	-	-
current	int	-	-
total	int	-	-

Returns: None

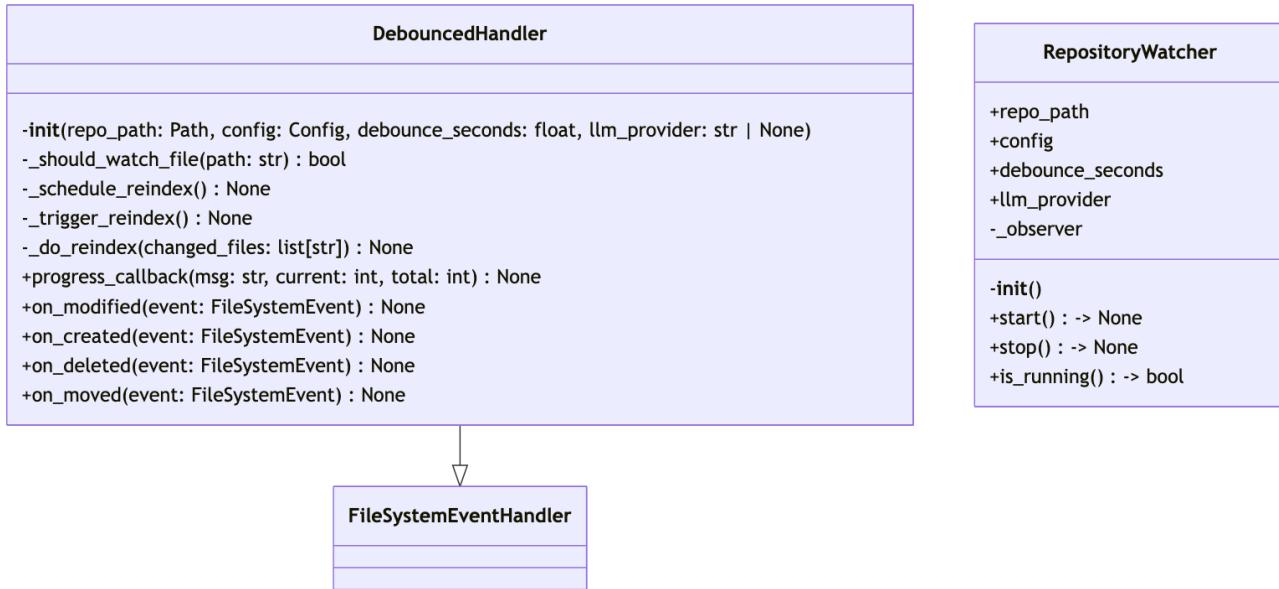
main

```
def main() -> None
```

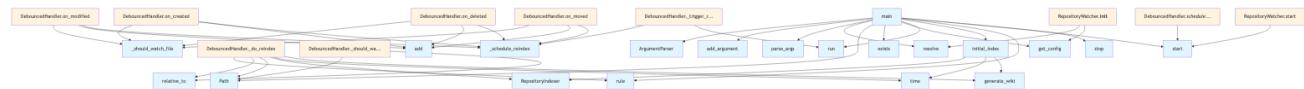
Main entry point for the watch command.

Returns: None

Class Diagram



Call Graph



Relevant Source Files

- src/local_deepwiki/watcher.py:29–223

Parser Module

File Overview

The parser module provides a unified interface for parsing source code files across multiple programming languages using tree-sitter parsers. It handles language detection, AST parsing, and extraction of code elements like docstrings and comments.

Classes

CodeParser

The CodeParser class serves as the `main` entry point for parsing source code files. It manages tree-sitter parsers for different programming languages and provides methods to parse files or source code strings.

Key Methods: - `__init__()` : Initializes the parser with empty dictionaries for parsers and languages - `_get_parser(language)` : Retrieves or creates a parser for the specified language - `detect_language()` : Detects the programming language of a file (method signature not shown in provided code) - `parse_file()` : Parses a source code file (method signature not shown in provided code) - `parse_source(source, language)` : Parses source code string and returns the AST root node - `get_file_info()` : Retrieves file information (method signature not shown in provided code)

Functions

get_node_text

Extracts text content from a tree-sitter node.

Parameters: - `node` (Node): The tree-sitter node - `source` (bytes): The original source bytes

Returns: - `str` : The text content of the node

get_docstring

Extracts docstring from a function or class node for different programming languages.

Parameters: - `node` (Node): The tree-sitter node - `source` (bytes): The original source bytes - `language` (LangEnum): The programming language

Returns: - `str | None` : The docstring or None if not found

The function handles language-specific docstring extraction, with visible support for Python docstrings that appear as the first expression statement in a function or class body.

Usage Examples

Basic Parser Usage

```
from local_deepwiki.core.parser import CodeParser
from local_deepwiki.core.models import LangEnum

# Initialize parser
parser = CodeParser()

# Parse source code string
source_code = "def hello():\n    pass"
ast_root = parser.parse_source(source_code, LangEnum.PYTHON)
```

Text Extraction

```
from local_deepwiki.core.parser import get_node_text

# Extract text from a node
node_text = get_node_text(node, source_bytes)
```

Docstring Extraction

```
from local_deepwiki.core.parser import get_docstring
from local_deepwiki.core.models import LangEnum

# Extract docstring from a function node
docstring = get_docstring(function_node, source_bytes, LangEnum.PYTHON)
```

Related Components

The parser module works with several other components:

- **LangEnum**: Enumeration for supported programming languages
- **Node**: Tree-sitter node objects representing AST elements
- **Parser**: Tree-sitter parser instances
- **Language**: Tree-sitter language definitions

The module imports multiple tree-sitter language modules including support for C, C#, C++, Go, Java, JavaScript, and others, indicating broad language support capabilities.

Implementation Notes

- The CodeParser class uses lazy initialization, creating parsers only when needed for specific languages
- Source code is handled as bytes internally, with UTF-8 encoding/decoding as needed
- The module includes special handling for PHP language module differences
- Error handling includes graceful fallback for text decoding issues

API Reference

class CodeParser

Multi-language code parser using tree-sitter.

Methods:**__init__**

```
def __init__()
```

Initialize the parser with language support.

detect_language

```
def detect_language(file_path: Path) -> LangEnum | None
```

Detect the programming language from file extension.

Parameter	Type	Default	Description
file_path	Path	-	Path to the source file.

parse_file

```
def parse_file(file_path: Path) -> tuple[Node, LangEnum, bytes] | None
```

Parse a source file and return the AST root.

Parameter	Type	Default	Description
file_path	Path	-	Path to the source file.

parse_source

```
def parse_source(source: str | bytes, language: LangEnum) -> Node
```

Parse source code string and return the AST root.

Parameter	Type	Default	Description
source	str bytes	-	The source code.
language	LangEnum	-	The programming language.

get_file_info

```
def get_file_info(file_path: Path, repo_root: Path) -> FileInfo
```

Get information about a source file. Uses chunked reading for large files to avoid loading the entire file into memory just for hash computation.

Parameter	Type	Default	Description
file_path	Path	-	Absolute path to the file.
repo_root	Path	-	Root directory of the repository.

Functions

get_node_text

```
def get_node_text(node: Node, source: bytes) -> str
```

Extract text content from a tree-sitter node.

Parameter	Type	Default	Description
node	Node	-	The tree-sitter node.
source	bytes	-	The original source bytes.

Returns: str

find_nodes_by_type

```
def find_nodes_by_type(root: Node, node_types: set[str]) -> list[Node]
```

Find all nodes of specified types in the AST.

Parameter	Type	Default	Description
root	Node	-	The root node to search from.
node_types	set[str]	-	Set of node type names to find .

Returns: list[Node]

walk

```
def walk(node: Node)
```

Parameter	Type	Default	Description
node	Node	-	-

get_node_name

```
def get_node_name(node: Node, source: bytes, language: LangEnum) -> str | None
```

Extract the name from a function/class/method node.

Parameter	Type	Default	Description
node	Node	-	The tree-sitter node.
source	bytes	-	The original source bytes.
language	LangEnum	-	The programming language.

Returns: str | None

get_docstring

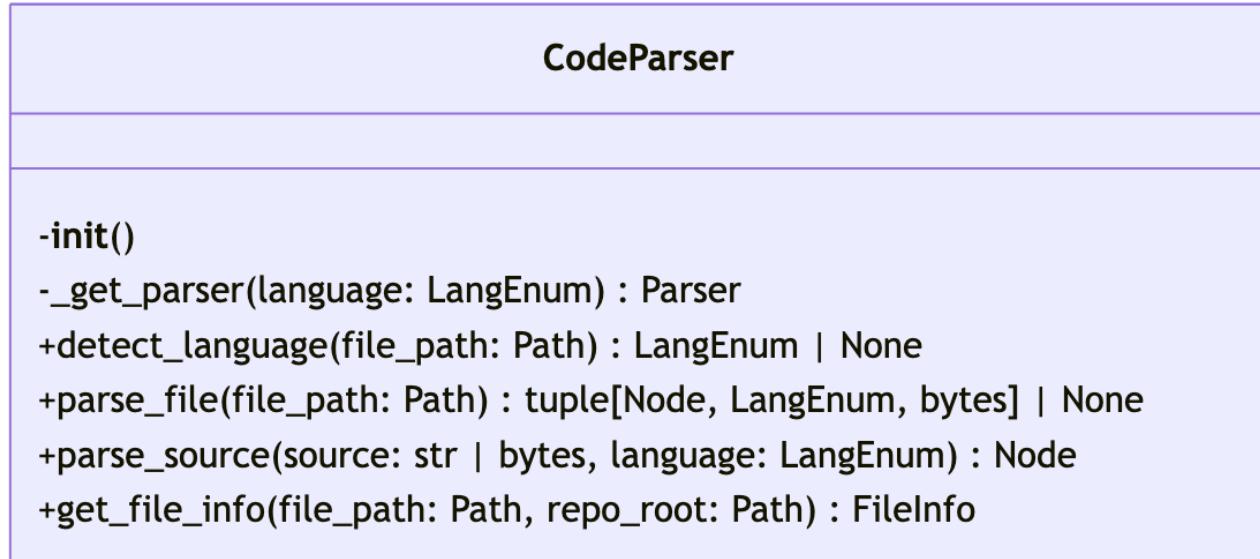
```
def get_docstring(node: Node, source: bytes, language: LangEnum) -> str | None
```

Extract docstring from a function/class node.

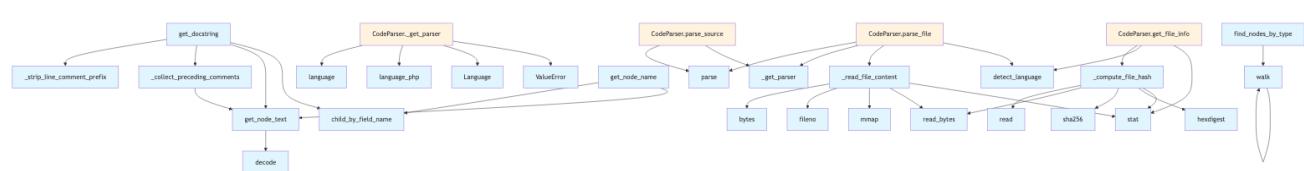
Parameter	Type	Default	Description
node	Node	-	The tree-sitter node.
source	bytes	-	The original source bytes.
language	LangEnum	-	The programming language.

Returns: str | None

Class Diagram



Call Graph



Relevant Source Files

- `src/local_deepwiki/core/parser.py:138–247`

See Also

- [test_api_docs](#) - uses this
- [test_parser](#) - uses this
- [api_docs](#) - uses this

vectorstore.py

File Overview

This module provides vector storage functionality for the local_deepwiki system using LanceDB as the underlying vector database. The VectorStore class handles embedding storage, similarity search, and retrieval operations for code chunks and documentation content.

Classes

VectorStore

The VectorStore class manages vector embeddings storage and retrieval operations using LanceDB. It provides functionality to store code chunks with their embeddings and perform similarity searches.

Key Dependencies: - Uses LanceDB for vector storage operations - Integrates with EmbeddingProvider for generating embeddings - Works with CodeChunk and SearchResult models for data handling

Functions

_sanitize_string_value

A utility function for sanitizing string values, likely used for data preprocessing before storage operations.

Related Components

This module integrates with several other components of the local_deepwiki system:

- **EmbeddingProvider:** Used for generating vector embeddings

- **CodeChunk**: Data model for code content that gets stored
- **SearchResult**: Data model for search operation results
- **ChunkType and Language**: Enums for categorizing content types
- **Logging system**: Uses the local_deepwiki logging utilities

The module serves as a core component for the vector storage layer, enabling semantic search capabilities across code documentation and chunks.

API Reference

class VectorStore

Vector store using LanceDB for code chunk storage and semantic search.

Methods:

__init__

```
def __init__(db_path: Path, embedding_provider: EmbeddingProvider)
```

Initialize the vector store.

Parameter	Type	Default	Description
db_path	Path	-	Path to the LanceDB database directory.
embedding_provider	EmbeddingProvider	-	Provider for generating embeddings.

create_or_update_table

```
async def create_or_update_table(chunks: list[CodeChunk]) -> int
```

Create or update the vector table with code chunks.

Parameter	Type	Default	Description
chunks	list[CodeChunk]	-	List of code chunks to store.

add_chunks

```
async def add_chunks(chunks: list[CodeChunk]) -> int
```

Add chunks to existing table.

Parameter	Type	Default	Description
chunks	list[CodeChunk]	-	List of code chunks to add.

search

```
async def search(query: str, limit: int = 10, language: str | None = None, chunk_type: str | None = None) -> list[CodeChunk]
```

Search for similar code chunks.

Parameter	Type	Default	Description
query	str	-	Search query text.
limit	int	10	Maximum number of results.
language	str None	None	Optional language filter.
chunk_type	str None	None	Optional chunk type filter.

get_chunk_by_id

```
async def get_chunk_by_id(chunk_id: str) -> CodeChunk | None
```

Get a specific chunk by ID.

Parameter	Type	Default	Description
chunk_id	str	-	The chunk ID.

get_chunks_by_file

```
async def get_chunks_by_file(file_path: str) -> list[CodeChunk]
```

Get all chunks for a specific file.

Parameter	Type	Default	Description
file_path	str	-	The file path.

delete_chunks_by_file

```
async def delete_chunks_by_file(file_path: str) -> int
```

Delete all chunks for a specific file.

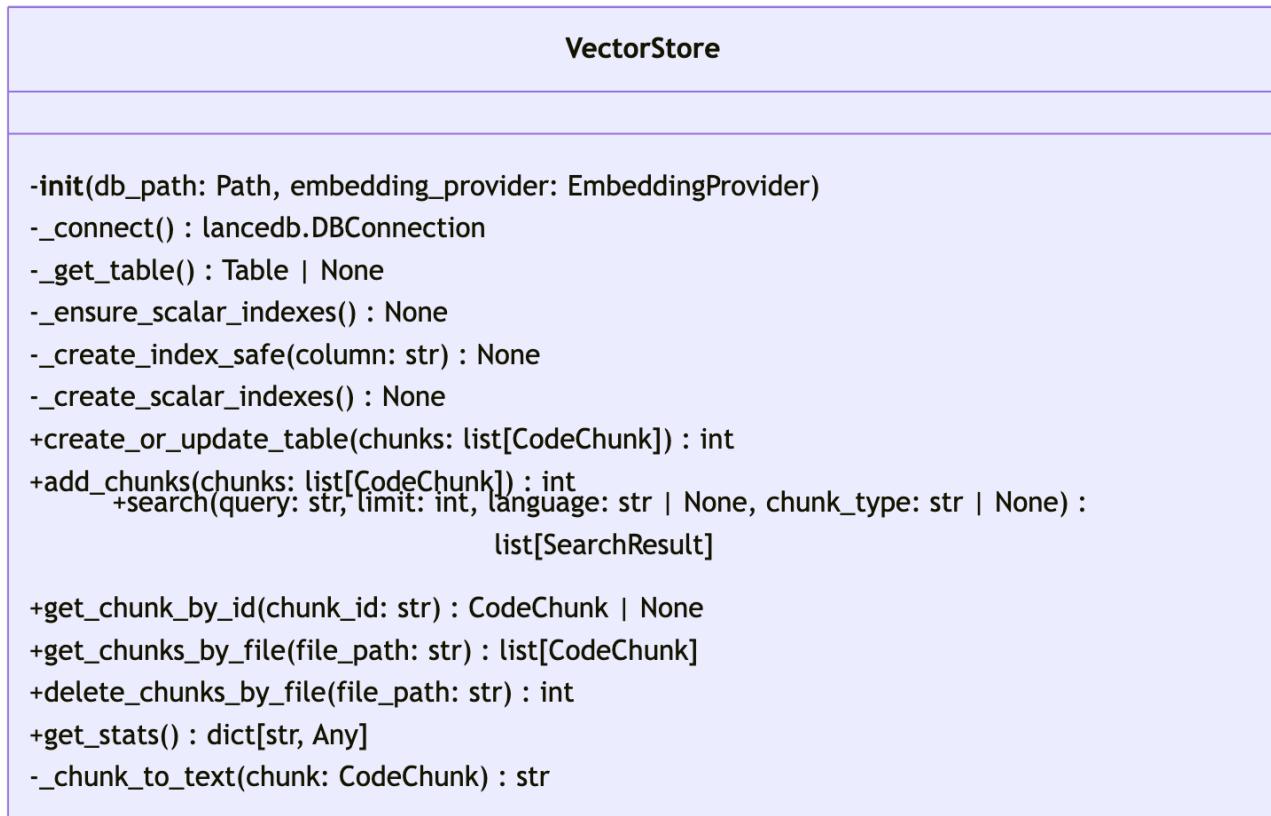
Parameter	Type	Default	Description
file_path	str	-	The file path.

get_stats

```
def get_stats() -> dict[str, Any]
```

Get statistics about the vector store.

Class Diagram



Call Graph



Relevant Source Files

- src/local_deepwiki/core/vectorstore.py:37-395

See Also

- [test_vectorstore](#) - uses this

- **server** - uses this

PDF Export Module

File Overview

The `pdf.py` module provides functionality for exporting DeepWiki documentation to PDF format. It supports both single-file exports (combining all pages) and separate exports (one PDF per page), with Mermaid diagram rendering capabilities and proper table of contents handling.

Classes

PdfExporter

The main class responsible for handling PDF export operations from DeepWiki documentation.

Initialization: - `wiki_path` : Path to the `.deepwiki` directory - `output_path` : Output path for PDF file(s) - `toc_entries` : List to store table of contents entries

Key Methods:

`export_single`

Exports all wiki pages to a single combined PDF file.

Returns: Path to the generated PDF file

This method loads the table of contents from `toc.json` if available and collects all pages in the proper order before generating the combined PDF.

`export_separate`

Exports each wiki page as a separate PDF file.

Returns: List of paths to generated PDF files

Creates an output directory and generates individual PDF files for each markdown file found in the wiki directory structure.

Functions

export_to_pdf

The main export function that provides a high-level interface for PDF generation.

Parameters: - `wiki_path` : Path to the .deepwiki directory (Path | str) - `output_path` : Output path, defaults to "wiki.pdf" or "wiki_pdfs/" (Path | str | None) - `single_file` : If True, combine all pages into one PDF (bool)

Returns: Success message with output path (str)

Raises: ValueError if the wiki path doesn't exist

main

CLI entry point for PDF export functionality. Provides command-line argument parsing for: -

`wiki_path` : Path to .deepwiki directory (defaults to ".deepwiki") - `--o`, `--output` : Output path specification - `--separate` : Flag for separate file export mode

Utility Functions

The module includes several utility functions for processing content:

- `is_mmdc_available` : Checks for Mermaid CLI availability
- `render_mermaid_to_png` / `render_mermaid_to_svg` : Mermaid diagram rendering
- `extract_mermaid_blocks` : Extracts Mermaid diagrams from markdown
- `render_markdown_for_pdf` : Processes markdown content for PDF generation
- `extract_title` : Extracts page titles from content

Usage Examples

Basic Single File Export

```
from pathlib import Path
from local_deepwiki.export.pdf import export_to_pdf

# Export all pages to a single PDF
result = export_to_pdf(
    wiki_path=Path(".deepwiki"),
    output_path=Path("documentation.pdf"),
    single_file=True
)
```

Separate File Export

```
# Export each page as a separate PDF
result = export_to_pdf(
    wiki_path=Path(".deepwiki"),
    output_path=Path("pdf_output"),
    single_file=False
)
```

Using the PdfExporter Class

```
from local_deepwiki.export.pdf import PdfExporter

exporter = PdfExporter(
    wiki_path=Path(".deepwiki"),
    output_path=Path("output.pdf")
)

# Single file export
pdf_path = exporter.export_single()

# Or separate files
pdf_paths = exporter.export_separate()
```

Dependencies

The module relies on: - `markdown` : For markdown processing - Standard library modules: `argparse`, `json`, `pathlib`, `subprocess`, `tempfile` - External tools: Mermaid CLI for diagram rendering (optional)

Related Components

This module integrates with the broader DeepWiki system by: - Reading from `.deepwiki` directory structures - Processing `toc.json` files for content organization - Working with markdown files containing documentation content

API Reference

`class PdfExporter`

Export wiki markdown to PDF format.

Methods:

`__init__`

```
def __init__(wiki_path: Path, output_path: Path)
```

Initialize the exporter.

Parameter	Type	Default	Description
<code>wiki_path</code>	<code>Path</code>	-	Path to the <code>.deepwiki</code> directory.
<code>output_path</code>	<code>Path</code>	-	Output path for PDF file(s).

`export_single`

```
def export_single() -> Path
```

Export all wiki pages to a single PDF.

export_separate

```
def export_separate() -> list[Path]
```

Export each wiki page as a separate PDF.

Functions

is_mmdc_available

```
def is_mmdc_available() -> bool
```

Check if mermaid-cli (mmdc) is available on the system.

Returns: bool

render_mermaid_to_png

```
def render_mermaid_to_png(diagram_code: str, timeout: int = 30) -> bytes | None
```

Render a mermaid diagram to PNG using mermaid-cli.

Parameter	Type	Default	Description
diagram_code	str	-	The mermaid diagram code.
timeout	int	30	Timeout in seconds for the mmdc command.

Returns: bytes | None

render_mermaid_to_svg

```
def render_mermaid_to_svg(diagram_code: str, timeout: int = 30) -> str | None
```

Render a mermaid diagram to SVG using mermaid-cli. Note: SVG may have font issues in PDF. Use render_mermaid_to_png for PDF export.

Parameter	Type	Default	Description
diagram_code	str	-	The mermaid diagram code.
timeout	int	30	Timeout in seconds for the mmdc command.

Returns: str | None

extract_mermaid_blocks

```
def extract_mermaid_blocks(content: str) -> list[tuple[str, str]]
```

Extract mermaid code blocks from markdown content.

Parameter	Type	Default	Description
content	str	-	Markdown content.

Returns: list[tuple[str, str]]

render_markdown_for_pdf

```
def render_markdown_for_pdf(content: str, render_mermaid: bool = True) -> str
```

Render markdown to HTML suitable for PDF.

Parameter	Type	Default	Description
content	str	-	Markdown content.
render_mermaid	bool	True	If True, attempt to render mermaid diagrams using CLI. Falls back to placeholder if CLI is not available.

Returns: str

extract_title

```
def extract_title(md_file: Path) -> str
```

Extract title from markdown file.

Parameter	Type	Default	Description
md_file	Path	-	Path to markdown file.

Returns: str

export_to_pdf

```
def export_to_pdf(wiki_path: Path | str, output_path: Path | str | None = None, single_file: bool = True)
```

Export wiki to PDF format.

Parameter	Type	Default	Description
wiki_path	Path str	-	Path to the .deepwiki directory.
output_path	Path str None	None	Output path (default: wiki.pdf or wiki_pdfs/).
single_file	bool	True	If True, combine all pages into one PDF.

Returns: str

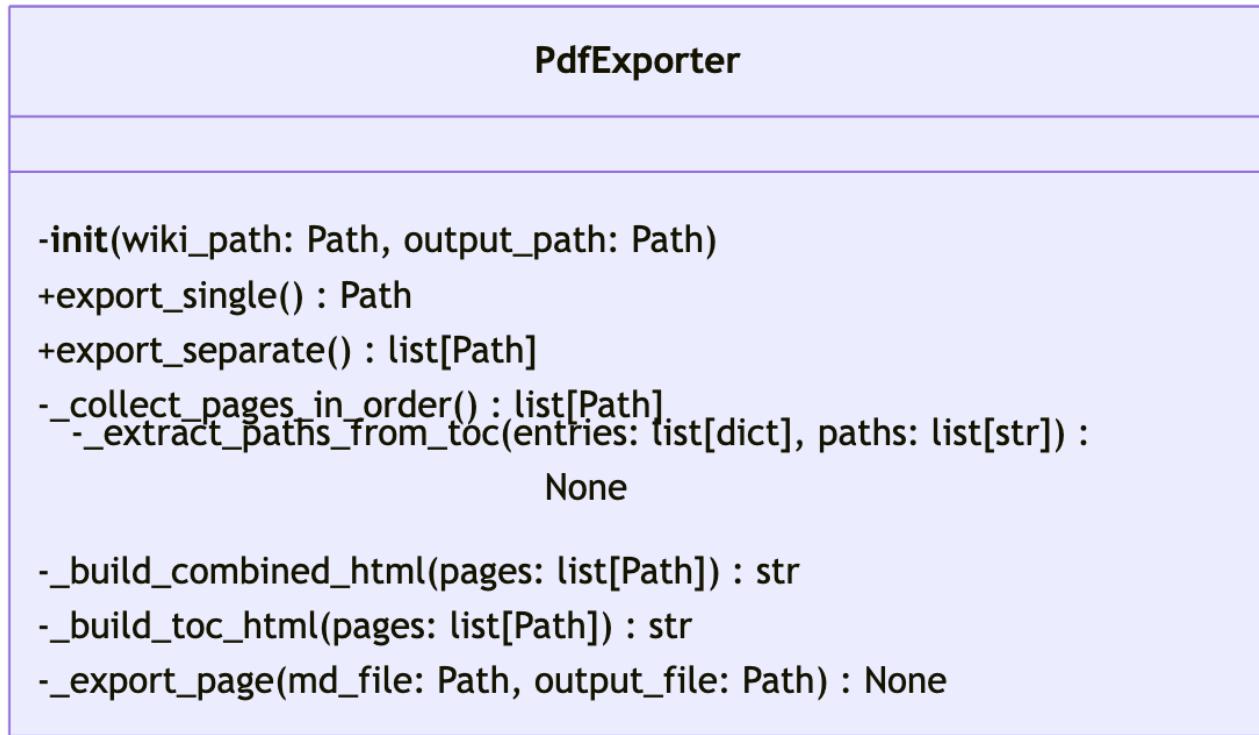
main

```
def main() -> None
```

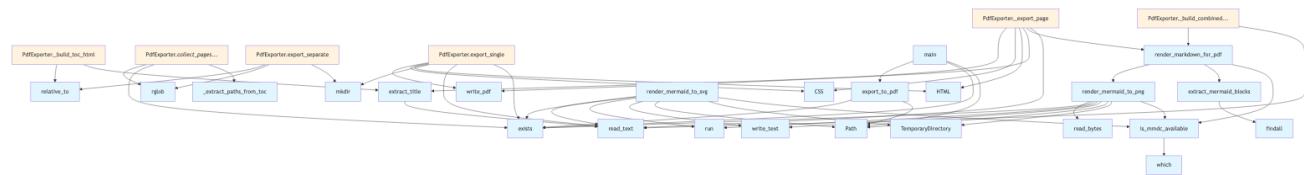
CLI entry point for PDF export.

Returns: None

Class Diagram



Call Graph



Relevant Source Files

- src/local_deepwiki/export/pdf.py:480–664

See Also

- [test_pdf_export](#) - uses this
- [server](#) - uses this
- [manifest](#) - shares 4 dependencies
- [vectorstore](#) - shares 3 dependencies

API Documentation Generator

File Overview

The `api_docs.py` file provides functionality for extracting and processing API documentation from Python code. It analyzes Python source code using tree-sitter parsing to extract function signatures, class definitions, docstrings, and other documentation-relevant information. The module supports parsing various docstring formats including Google and NumPy styles.

Classes

Parameter

A dataclass that represents a function or method parameter.

Fields: - `name` : The parameter name - `type_hint` : Optional type annotation for the parameter
- `default_value` : Optional default value if the parameter has one - `description` : Optional description extracted from docstring

FunctionSignature

A dataclass that captures the complete signature and documentation of a function or method.

Fields: - `name` : The function name - `parameters` : List of Parameter objects representing the function arguments - `return_type` : Optional return type annotation - `decorators` : List of decorator names applied to the function - `docstring` : The raw docstring text - `parsed_docstring` : Dictionary containing parsed docstring components - `line_number` : Line number where the function is defined - `is_async` : Boolean indicating if this is an async function - `is_method` : Boolean indicating if this is a class method - `is_property` : Boolean indicating if this is a property - `visibility` : String indicating visibility (public, private, protected)

ClassSignature

A dataclass that represents a class definition with its methods and documentation.

Fields: - `name` : The class name - `base_classes` : List of base class names this class inherits from - `methods` : List of FunctionSignature objects for class methods - `properties` : List of FunctionSignature objects for class properties - `docstring` : The raw class docstring - `parsed_docstring` : Dictionary containing parsed docstring components - `line_number` : Line number where the class is defined - `decorators` : List of decorator names applied to the class - `visibility` : String indicating visibility (public, private, protected)

APIDocExtractor

The `main` class responsible for extracting API documentation from Python source code.

Methods: - `extract_from_file` : Extracts documentation from a single Python file - `extract_from_content` : Extracts documentation from Python source code content - `_extract_functions` : Internal method to extract function signatures - `_extract_classes` : Internal method to extract class signatures

Functions

extract_python_parameters

Extracts parameter information from a Python function's parameter list node.

Parameters: - `params_node` : Tree-sitter node representing the function parameters - `source_code` : The source code string for context

Returns: List of Parameter objects

extract_python_return_type

Extracts the return type annotation from a Python function node.

Parameters: - `func_node` : Tree-sitter node representing the function - `source_code` : The source code string for context

Returns: Optional string containing the return type

extract_python_decorators

Extracts decorator names from a Python function or class node.

Parameters: - `node` : Tree-sitter node representing the decorated item - `source_code` : The source code string for context

Returns: List of decorator name strings

extract_python_docstring

Extracts the docstring from a Python function or class node.

Parameters: - `node` : Tree-sitter node representing the function or class - `source_code` : The source code string for context

Returns: Optional string containing the docstring content

parse_google_docstring

Parses a Google-style docstring into structured components.

Parameters: - `docstring` : The raw docstring text to parse

Returns: Dictionary with parsed sections like args, returns, raises, etc.

parse_numpy_docstring

Parses a NumPy-style docstring into structured components.

Parameters: - `docstring` : The raw docstring text to parse

Returns: Dictionary with parsed sections following NumPy documentation format

parse_docstring

General docstring parser that attempts to detect the format and parse accordingly.

Parameters: - `docstring` : The raw docstring text to parse

Returns: Dictionary with parsed docstring components

Usage Examples

Extracting API Documentation from a File

```
from local_deepwiki.generators.api_docs import APIDocExtractor

# Create an extractor instance
extractor = APIDocExtractor()

# Extract documentation from a Python file
file_path = Path("example.py")
functions, classes = extractor.extract_from_file(file_path)

# Process the extracted functions
for func in functions:
    print(f"Function: {func.name}")
    print(f"Parameters: {[p.name for p in func.parameters]}")
    print(f"Return type: {func.return_type}")
```

Working with Function Signatures

```
# Create a parameter
param = Parameter(
    name="value",
    type_hint="str",
    default_value=None,
    description="Input value to process"
)

# Access function signature details
if func.is_async:
    print(f"Async function: {func.name}")

if func.decorators:
    print(f"Decorators: {func.decorators}")
```

Parsing Docstrings

```
from local_deepwiki.generators.api_docs import parse_docstring

docstring = """
Process the input data.

Args:
    data: The input data to process
    options: Optional processing parameters

Returns:
    Processed data result
"""

parsed = parse_docstring(docstring)
print(parsed['args']) # Parameter descriptions
print(parsed['returns']) # Return value description
```

Related Components

This module integrates with several other components from the local_deepwiki package:

- **CodeParser**: Used for parsing Python source code with tree-sitter
- **Language**: Enum defining supported programming languages
- **Core chunker**: Provides node type constants for classes and functions
- **Parser utilities**: Functions like `find_nodes_by_type`, `get_node_name`, and `get_node_text` for tree-sitter node manipulation

The module works with tree-sitter Node objects to analyze Python syntax trees and extract structured documentation information.

API Reference

`class Parameter`

Represents a function parameter.

class FunctionSignature

Represents a function/method signature.

class ClassSignature

Represents a class signature.

class APIDocExtractor

Extracts API documentation from source files.

Methods:

`__init__`

```
def __init__()
```

Initialize the extractor.

`extract_from_file`

```
def extract_from_file(file_path: Path) -> tuple[list[FunctionSignature], list[ClassSignature]]
```

Extract API documentation from a source file.

Parameter	Type	Default	Description
file_path	Path	-	Path to the source file.

Functions

`extract_python_parameters`

```
def extract_python_parameters(func_node: Node, source: bytes) -> list[Parameter]
```

Extract parameters from a Python function definition.

Parameter	Type	Default	Description
func_node	Node	-	The function_definition AST node.
source	bytes	-	Source code bytes.

Returns: list[Parameter]

extract_python_return_type

```
def extract_python_return_type(func_node: Node, source: bytes) -> str | None
```

Extract return type annotation from a Python function.

Parameter	Type	Default	Description
func_node	Node	-	The function_definition AST node.
source	bytes	-	Source code bytes.

Returns: str | None

extract_python_decorators

```
def extract_python_decorators(func_node: Node, source: bytes) -> list[str]
```

Extract decorators from a Python function.

Parameter	Type	Default	Description
func_node	Node	-	The function_definition AST node.
source	bytes	-	Source code bytes.

Returns: list[str]

extract_python_docstring

```
def extract_python_docstring(node: Node, source: bytes) -> str | None
```

Extract docstring from a Python function or class.

Parameter	Type	Default	Description
node	Node	-	The function_definition or class_definition AST node.
source	bytes	-	Source code bytes.

Returns: str | None

parse_google_docstring

```
def parse_google_docstring(docstring: str) -> dict
```

Parse a Google-style docstring.

Parameter	Type	Default	Description
docstring	str	-	The docstring content.

Returns: dict

parse_numpy_docstring

```
def parse_numpy_docstring(docstring: str) -> dict
```

Parse a NumPy-style docstring.

Parameter	Type	Default	Description
docstring	str	-	The docstring content.

Returns: dict

parse_docstring

```
def parse_docstring(docstring: str) -> dict
```

Parse a docstring, auto-detecting format.

Parameter	Type	Default	Description
docstring	str	-	The docstring content.

Returns: dict

extract_function_signature

```
def extract_function_signature(func_node: Node, source: bytes, language: Language, class_name: str | None) -> FunctionSignature | None
```

Extract signature from a function node.

Parameter	Type	Default	Description
func_node	Node	-	The function AST node.
source	bytes	-	Source code bytes.
language	Language	-	Programming language.
class_name	str None	None	Parent class name if this is a method.

Returns: FunctionSignature | None

extract_class_signature

```
def extract_class_signature(class_node: Node, source: bytes, language: Language) -> ClassSignature | None
```

Extract signature from a class node.

Parameter	Type	Default	Description
class_node	Node	-	The class AST node.
source	bytes	-	Source code bytes.
language	Language	-	Programming language.

Returns: ClassSignature | None

format_parameter

```
def format_parameter(param: Parameter) -> str
```

Format a parameter for display.

Parameter	Type	Default	Description
param	Parameter	-	The parameter to format.

Returns: str

format_function_signature_line

```
def format_function_signature_line(sig: FunctionSignature) -> str
```

Format a function signature as a single line.

Parameter	Type	Default	Description
sig	FunctionSignature	-	The function signature.

Returns: str

generate_api_reference_markdown

```
def generate_api_reference_markdown(functions: list[FunctionSignature], classes: list[ClassSignature]) -> str
```

Generate markdown API reference documentation.

Parameter	Type	Default	Description
functions	list[FunctionSignature]	-	List of function signatures.
classes	list[ClassSignature]	-	List of class signatures.
include_private	bool	False	Whether to include private (underscore) items.

Returns: str

get_file_api_docs

```
def get_file_api_docs(file_path: Path) -> str | None
```

Get API documentation for a single file.

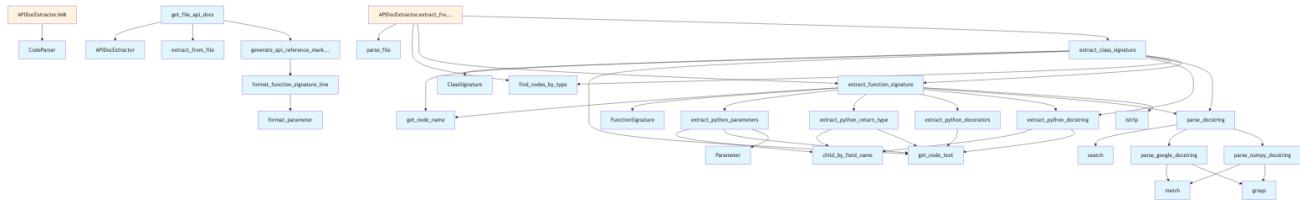
Parameter	Type	Default	Description
file_path	Path	-	Path to the source file.

Returns: str | None

Class Diagram



Call Graph



Relevant Source Files

- `src/local_deepwiki/generators/api_docs.py:15–21`

See Also

- [test_api_docs](#) - uses this
- [crosslinks](#) - shares 4 dependencies
- [diagrams](#) - shares 4 dependencies

CrossLinks Module

Overview

The crosslinks module provides functionality for automatically adding cross-links between wiki pages. It processes wiki content to identify entity references (like class names, function names, etc.) and converts them into clickable links to their corresponding documentation pages.

Classes

EntityInfo

A data class that stores information about documented entities.

EntityRegistry

A registry that maintains information about all documented entities that can be cross-linked.

CrossLinker

The [main](#) class responsible for adding cross-links to wiki pages.

Methods

- `__init__` : Initializes the CrossLinker with an EntityRegistry
- `add_links(page: WikiPage) -> WikiPage` : Adds cross-links to a wiki page and returns a new WikiPage instance with the processed content
- `_process_content` : Internal method for processing page content
- `_split_by_code_blocks` : Internal method for handling code block sections
- `_add_links_to_text` : Internal method for adding links to text content
- `_replace_entity_mentions` : Internal method for replacing entity mentions with links
- `protect` : Internal method for protecting certain content from link processing
- `_link_backticked_entities` : Converts backticked entity names to markdown links

- `qualified_replacement` : Internal method for handling qualified entity names
- `_relative_path` : Calculates relative paths between wiki pages

Functions

`add_cross_links`

```
def add_cross_links(  
    pages: list[WikiPage],  
    registry: EntityRegistry,  
) -> list[WikiPage]:
```

Processes a list of wiki pages to add cross-links to all of them.

Parameters: - `pages` : List of WikiPage objects to process - `registry` : EntityRegistry containing documented entities

Returns: - List of WikiPage objects with cross-links added

Usage Examples

Basic Cross-Linking

```
from local_deepwiki.generators.crosslinks import add_cross_links  
  
# Process multiple pages  
linked_pages = add_cross_links(wiki_pages, entity_registry)
```

Single Page Processing

```
from local_deepwiki.generators.crosslinks import CrossLinker

# Create a linker instance
linker = CrossLinker(registry)

# Process a single page
linked_page = linker.add_links(original_page)
```

Key Features

Backticked Entity Linking

The CrossLinker can handle various formats of backticked entity references:

- `EntityName` → [EntityName](path)
- `module.EntityName` → [EntityName](path)
- `module.submodule.EntityName` → [EntityName](path)

PROTECTED5

The `_relative_path` method calculates proper relative paths between wiki pages, handling different directory structures like: - Source: "modules/src.md" - Target: "files/src/indexer.md"

PROTECTED6

This module works with:

- **WikiPage**: The data model representing wiki pages (from `local_deepwiki.models`)
- **CodeChunk** and **ChunkType**: Used for processing code content (from `local_deepwiki.models`)

The module integrates into the larger wiki generation pipeline by taking processed wiki pages and enhancing them with cross-references to create a more navigable documentation structure.

API Reference

class EntityInfo

Information about a documented entity.

class EntityRegistry

Registry of documented entities and their wiki page locations. This class maintains a mapping of entity names (classes, functions, etc.) to their documentation page paths, enabling cross-linking between pages.

Methods:

__init__

```
def __init__() -> None
```

Initialize an empty entity registry.

register_entity

```
def register_entity(name: str, entity_type: ChunkType, wiki_path: str, file_path: str, parent_name: ...)
```

Register a documented entity.

Parameter	Type	Default	Description
name	str	-	The entity name (e.g., " WikiGenerator ").
entity_type	ChunkType	-	The type of entity (class, function, etc.).
wiki_path	str	-	Path to the wiki page documenting this entity.
file_path	str	-	Path to the source file containing this entity.
parent_name	str None	None	Parent entity name (e.g., class name for methods).

register_from_chunks

```
def register_from_chunks(chunks: list[CodeChunk], wiki_path: str) -> None
```

Register entities from a list of code chunks.

Parameter	Type	Default	Description
chunks	list[CodeChunk]	-	List of code chunks from a file.
wiki_path	str	-	Path to the wiki page for these chunks.

get_entity

```
def get_entity(name: str) -> EntityInfo | None
```

Get entity info by name.

Parameter	Type	Default	Description
name	str	-	The entity name to look up.

get_entity_by_alias

```
def get_entity_by_alias(alias: str) -> tuple[str, EntityInfo] | None
```

Get entity info by alias (spaced name).

Parameter	Type	Default	Description
alias	str	-	The spaced alias to look up (e.g., "Vector Store").

get_all_aliases

```
def get_all_aliases() -> dict[str, str]
```

Get all registered aliases.

get_all_entities

```
def get_all_entities() -> dict[str, EntityInfo]
```

Get all registered entities.

get_page_entities

```
def get_page_entities(wiki_path: str) -> list[str]
```

Get all entities defined in a specific wiki page.

Parameter	Type	Default	Description
wiki_path	str	-	The wiki page path.

class CrossLinker

Adds cross-links to wiki page content. This class processes wiki page content and replaces mentions of documented entities with markdown links to their documentation pages.

Methods:**__init__**

```
def __init__(registry: EntityRegistry) -> None
```

Initialize the cross-linker.

Parameter	Type	Default	Description
registry	EntityRegistry	-	The entity registry to use for lookups.

add_links

```
def add_links(page: WikiPage) -> WikiPage
```

Add cross-links to a wiki page.

Parameter	Type	Default	Description
page	WikiPage	-	The wiki page to process.

protect

```
def protect(match: re.Match) -> str
```

Parameter	Type	Default	Description
match	re.Match	-	-

qualified_replacement

```
def qualified_replacement(match: re.Match) -> str
```

Parameter	Type	Default	Description
match	re.Match	-	-

Functions

camel_to_spaced

```
def camel_to_spaced(name: str) -> str | None
```

Convert CamelCase to 'Spaced Words'.

Parameter	Type	Default	Description
name	str	-	The CamelCase name.

Returns: str | None

add_cross_links

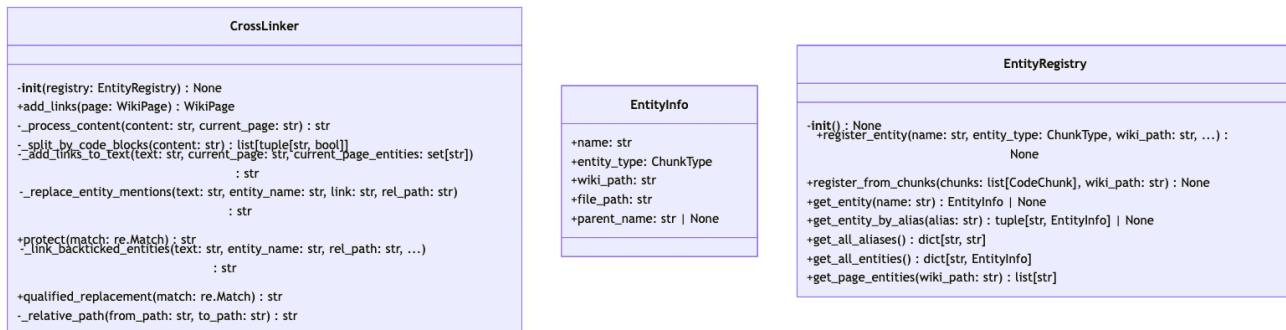
```
def add_cross_links(pages: list[WikiPage], registry: EntityRegistry) -> list[WikiPage]
```

Add cross-links to all wiki pages.

Parameter	Type	Default	Description
pages	list[WikiPage]	-	List of wiki pages to process.
registry	EntityRegistry	-	Entity registry with documented entities.

Returns: list[WikiPage]

Class Diagram



Call Graph



Relevant Source Files

- `src/local deepwiki/generators/crosslinks.py:16-23`

See Also

- [test_crosslinks](#) - uses this
- [diagrams](#) - shares 4 dependencies
- [api_docs](#) - shares 4 dependencies

diagrams.py

File Overview

The `diagrams.py` module provides functionality for generating various types of Mermaid diagrams from code analysis data. It creates visual representations including class diagrams, dependency graphs, module overviews, language distribution charts, and sequence diagrams to help understand code structure and relationships.

Classes

ClassInfo

A dataclass that stores information about a class for diagram generation purposes.

Attributes: - `name` : The class name - `methods` : List of method names - `attributes` : List of attribute names
- `parents` : List of parent class names - `is_abstract` : Boolean indicating if the class is abstract (default: False) - `is_dataclass` : Boolean indicating if the class is a dataclass (default: False) - `docstring` : Optional docstring content

Functions

sanitize_mermaid_name

```
def sanitize_mermaid_name(name: str) -> str
```

Sanitizes a name to make it safe for use in Mermaid diagram syntax by replacing problematic characters with underscores and ensuring the name starts with a letter.

Parameters: - `name` : Original name to sanitize

Returns: - Sanitized name safe for Mermaid syntax

generate_class_diagram

```
def generate_class_diagram(
    chunks: list,
    show_attributes: bool = True,
    show_types: bool = True,
    max_methods: int = 15,
) -> str | None
```

Generates an enhanced Mermaid class diagram from code chunks with support for showing class attributes, type annotations, inheritance relationships, and distinguishing special class types like abstract classes and dataclasses.

Parameters: - `chunks` : List of CodeChunk or SearchResult objects - `show_attributes` : Whether to display class attributes (default: True) - `show_types` : Whether to show type annotations (default: True) - `max_methods` : Maximum number of methods to display per class (default: 15)

Returns: - Mermaid class diagram string, or None if no classes found

generate_dependency_graph

```
def generate_dependency_graph(
    chunks: list,
    project_name: str = "project",
    detect_circular: bool = True,
) -> str | None
```

Creates a Mermaid flowchart showing module dependencies with optional circular dependency detection and highlighting.

Parameters: - `chunks` : List of CodeChunk objects (should include IMPORT chunks) - `project_name` : Name of the project for filtering internal imports (default: "project") - `detect_circular` : Whether to highlight circular dependencies (default: True)

Returns: - Mermaid flowchart markdown string, or None if no dependencies found

generate_module_overview

```
def generate_module_overview(
    index_status: IndexStatus,
    show_file_counts: bool = True,
) -> str | None
```

Generates a high-level module overview diagram showing package structure with subgraphs for major directories.

Parameters: - `index_status` : `IndexStatus` object containing file information - `show_file_counts` : Whether to show file counts in diagram nodes (default: True)

Returns: - Mermaid diagram string, or None if insufficient structure

generate_language_pie_chart

```
def generate_language_pie_chart(index_status: IndexStatus) -> str | None
```

Creates a pie chart showing the distribution of programming languages in the codebase.

Parameters: - `index_status` : `IndexStatus` object with language count data

Returns: - Mermaid pie chart string, or None if no languages found

generate_sequence_diagram

```
def generate_sequence_diagram(
    call_graph: dict[str, list[str]],
    entry_point: str | None = None,
    max_depth: int = 5,
) -> str | None
```

Generates a sequence diagram from a call graph showing the sequence of function calls starting from an entry point.

Parameters: - `call_graph` : Dictionary mapping caller functions to lists of callees - `entry_point` : Starting function name (if None, uses most-called function) - `max_depth` : Maximum call depth to display (default: 5)

Returns: - Mermaid sequence diagram string, or None if call graph is empty

_find_circular_dependencies

```
def _find_circular_dependencies(deps: dict[str, set[str]]) -> set[tuple[str, str]]
```

Internal function that finds circular dependencies in a dependency graph using depth-first search.

Parameters: - `deps` : Dictionary mapping modules to their dependency sets

Returns: - Set of (from, to) tuples representing circular dependency edges

collect_participants

```
def collect_participants(func: str, depth: int) -> None
```

Internal helper function for collecting participants in a sequence diagram by traversing the call graph recursively up to a maximum depth.

Parameters: - `func` : Function name to start collection from - `depth` : Current traversal depth

Usage Examples

Generating a Class Diagram

```
from local_deepwiki.generators.diagrams import generate_class_diagram

# Generate diagram with default settings
diagram = generate_class_diagram(code_chunks)

# Generate diagram without attributes but with type info
diagram = generate_class_diagram(
    code_chunks,
    show_attributes=False,
    show_types=True,
    max_methods=10
)
```

Creating a Dependency Graph

```
from local_deepwiki.generators.diagrams import generate_dependency_graph

# Generate dependency graph with circular detection
graph = generate_dependency_graph(
    import_chunks,
    project_name="my_project",
    detect_circular=True
)
```

Generating Language Distribution

```
from local_deepwiki.generators.diagrams import generate_language_pie_chart

# Create pie chart from index status
chart = generate_language_pie_chart(index_status)
```

Related Components

This module works with several other components from the local_deepwiki system:

- **CodeChunk**: Used as input for analyzing code structure and generating diagrams
- **IndexStatus**: Provides file and language statistics for overview diagrams
- **ChunkType**: Referenced for filtering specific types of code chunks (like IMPORT chunks)

The module imports from `local_deepwiki.models` to access these data structures and uses standard library modules like `re`, `dataclasses`, `pathlib`, and `typing` for its functionality.

API Reference

`class ClassInfo`

Information about a class for diagram generation.

Functions

`sanitize_mermaid_name`

```
def sanitize_mermaid_name(name: str) -> str
```

Sanitize a name for use in Mermaid diagrams.

Parameter	Type	Default	Description
name	str	-	Original name.

Returns: str

`generate_class_diagram`

```
def generate_class_diagram(chunks: list, show_attributes: bool = True, show_types: bool = True, max_
```

Generate an enhanced Mermaid class diagram from code chunks. Features:
- Shows class attributes/properties (not just methods)
- Shows type annotations for parameters and return types
- Distinguishes abstract classes, dataclasses, protocols
- Shows inheritance relationships

Parameter	Type	Default	Description
chunks	list	-	List of CodeChunk or SearchResult objects.
show_attributes	bool	True	Whether to show class attributes.
show_types	bool	True	Whether to show type annotations.
max_methods	int	15	Maximum methods to show per class.

Returns: str | None

`generate_dependency_graph`

```
def generate_dependency_graph(chunks: list, project_name: str = "project", detect_circular: bool = T
```

Generate a Mermaid flowchart showing module dependencies with circular detection.

Parameter	Type	Default	Description
chunks	list	-	List of CodeChunk objects (should include IMPORT chunks).
project_name	str	"project"	Name of the project for filtering internal imports.
detect_circular	bool	True	Whether to highlight circular dependencies.

Returns: str | None

dfs

```
def dfs(node: str, path: list[str], visited: set[str]) -> None
```

Parameter	Type	Default	Description
node	str	-	-
path	list[str]	-	-
visited	set[str]	-	-

Returns: None

generate_module_overview

```
def generate_module_overview(index_status: IndexStatus, show_file_counts: bool = True) -> str | None
```

Generate a high-level module overview diagram. Shows package structure with subgraphs for major directories.

Parameter	Type	Default	Description
index_status	IndexStatus	-	Index status with file information.
show_file_counts	bool	True	Whether to show file counts in nodes.

Returns: str | None**generate_language_pie_chart**

```
def generate_language_pie_chart(index_status: IndexStatus) -> str | None
```

Generate a pie chart showing language distribution.

Parameter	Type	Default	Description
index_status	IndexStatus	-	Index status with language counts.

Returns: str | None**generate_sequence_diagram**

```
def generate_sequence_diagram(call_graph: dict[str, list[str]], entry_point: str | None = None, max_
```

Generate a sequence diagram from a call graph. Shows the sequence of calls starting from an entry point.

Parameter	Type	Default	Description
call_graph	dict[str, list[str]]	-	Mapping of caller to list of callees.
entry_point	str None	None	Starting function (if None, uses most-called function).
max_depth	int	5	Maximum call depth to show.

Returns: str | None**collect_participants**

```
def collect_participants(func: str, depth: int) -> None
```

Parameter	Type	Default	Description
func	str	-	-
depth	int	-	-

Returns: None

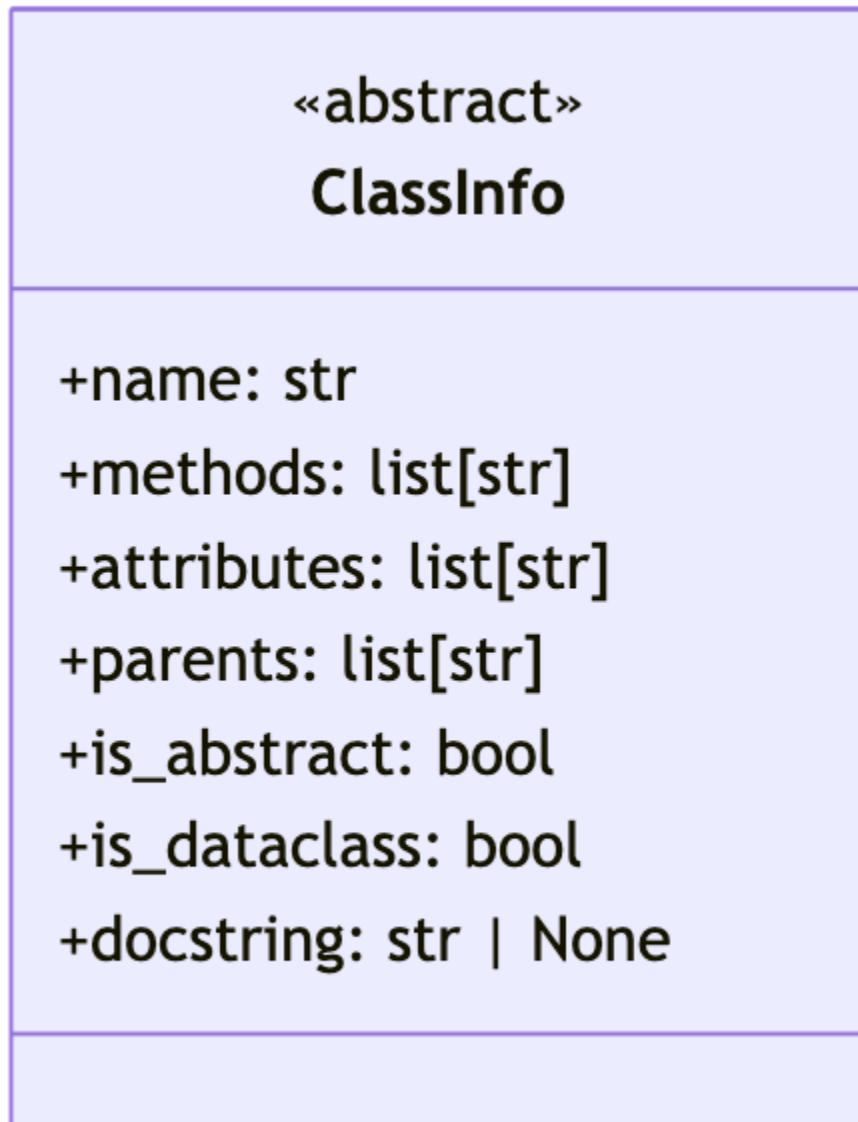
add_calls

```
def add_calls(caller: str, depth: int) -> None
```

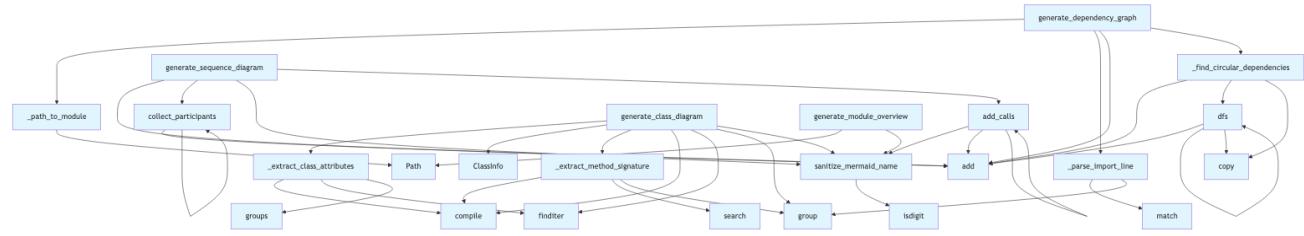
Parameter	Type	Default	Description
caller	str	-	-
depth	int	-	-

Returns: None

Class Diagram



Call Graph



Relevant Source Files

- `src/local_deepwiki/generators/diagrams.py:12-21`

See Also

- [crosslinks](#) - shares 4 dependencies
- [api_docs](#) - shares 4 dependencies

Manifest Generator Module

File Overview

The `manifest.py` module provides functionality for parsing and caching project manifest information from various file formats. It handles project metadata extraction from Python projects (`pyproject.toml`, `setup.py`, `requirements.txt`) and Node.js projects (`package.json`), with built-in caching mechanisms to improve performance.

Classes

ManifestCacheEntry

A dataclass that represents a cached manifest entry, storing both the manifest data and metadata for cache validation.

ProjectManifest

A dataclass that contains parsed project manifest information including dependencies, metadata, and project structure details.

Functions

Cache Management Functions

_get_manifest_mtimes Retrieves modification times for manifest files to determine if cache is still valid.

_is_cache_valid Validates whether the cached manifest data is still current by comparing file modification times.

_load_manifest_cache Loads cached manifest data from disk storage.

_save_manifest_cache Persists manifest data to cache storage for future use.

Serialization Functions

_manifest_to_dict Converts a ProjectManifest object to a dictionary format suitable for serialization.

_manifest_from_dict Reconstructs a ProjectManifest object from a dictionary representation.

Main Interface Functions

get_cached_manifest Primary function that retrieves manifest data, using cache when valid or parsing fresh when needed.

parse_manifest Parses project manifest files and returns a ProjectManifest object with extracted information.

Format-Specific Parsers

_parse_pyproject_toml Parses Python project configuration from pyproject.toml files using the `tomllib` library.

_parse_python_dep Extracts and normalizes Python dependency information.

_parse_setup_py Processes setup.py files to extract project metadata and dependencies.

_parse_requirements_txt Parses requirements.txt files for Python dependency lists.

_parse_package_json Handles Node.js package.json files for JavaScript/TypeScript project information.

Usage Examples

```
from local_deepwiki.generators.manifest import get_cached_manifest, parse_manifest
from pathlib import Path

# Get cached manifest (preferred method)
project_path = Path("./my_project")
manifest = get_cached_manifest(project_path)

# Parse manifest directly
manifest = parse_manifest(project_path)

# Access manifest data
print(f"Project dependencies: {manifest.dependencies}")
print(f"Project metadata: {manifest.metadata}")
```

Related Components

This module integrates with:

- **local_deepwiki.logging**: Uses the logging system via `get_logger` for operation tracking
- **pathlib.Path**: Extensively uses Path objects for file system operations
- **tomllib/tomli**: Handles TOML file parsing for Python project configurations
- **json**: Processes JSON files for Node.js projects and cache serialization

The module serves as a foundational component for project analysis, providing standardized manifest information that other parts of the deepwiki system can consume for documentation generation and project understanding.

API Reference

`class ManifestCacheEntry`

Cache entry storing manifest data and file modification times.

Methods:

to_dict

```
def to_dict() -> dict[str, Any]
```

Convert to dictionary for JSON serialization.

from_dict

```
def from_dict(data: dict[str, Any]) -> "ManifestCacheEntry"
```

Create from dictionary.

Parameter	Type	Default	Description
data	dict[str, Any]	-	-

class ProjectManifest

Extracted project metadata from package manifests.

Methods:**has_data**

```
def has_data() -> bool
```

Check if any meaningful data was extracted.

get_tech_stack_summary

```
def get_tech_stack_summary() -> str
```

Generate a factual tech stack summary.

get_dependency_list

```
def get_dependency_list() -> str
```

Get a formatted list of all dependencies.

get_entry_points_summary

```
def get_entry_points_summary() -> str
```

Get a summary of entry points and scripts.

Functions

get_cached_manifest

```
def get_cached_manifest(repo_path: Path, cache_dir: Path | None = None) -> ProjectManifest
```

Get project manifest, using cache if available and valid. This function checks if a cached manifest exists and is still valid (no manifest files have been modified). If valid, returns cached data. Otherwise, parses fresh and updates the cache.

Parameter	Type	Default	Description
repo_path	Path	-	Path to the repository root.
cache_dir	Path None	None	Directory for cache storage (defaults to repo_path/.deepwiki).

Returns: ProjectManifest

parse_manifest

```
def parse_manifest(repo_path: Path) -> ProjectManifest
```

Parse all recognized package manifests in a repository. Note: For incremental updates, prefer get_cached_manifest() which avoids re-parsing when manifest files haven't changed.

Parameter	Type	Default	Description
repo_path	Path	-	Path to the repository root.

Returns: ProjectManifest**find**

```
def find(path: str) -> Any
```

Parameter	Type	Default	Description
path	str	-	-

Returns: Any**get_directory_tree**

```
def get_directory_tree(repo_path: Path, max_depth: int = 3, max_items: int = 50) -> str
```

Generate a directory tree structure for the repository.

Parameter	Type	Default	Description
repo_path	Path	-	Path to repository root.
max_depth	int	3	Maximum depth to traverse.
max_items	int	50	Maximum total items to include.

Returns: str**should_skip**

```
def should_skip(name: str) -> bool
```

Parameter	Type	Default	Description
name	str	-	-

Returns: bool

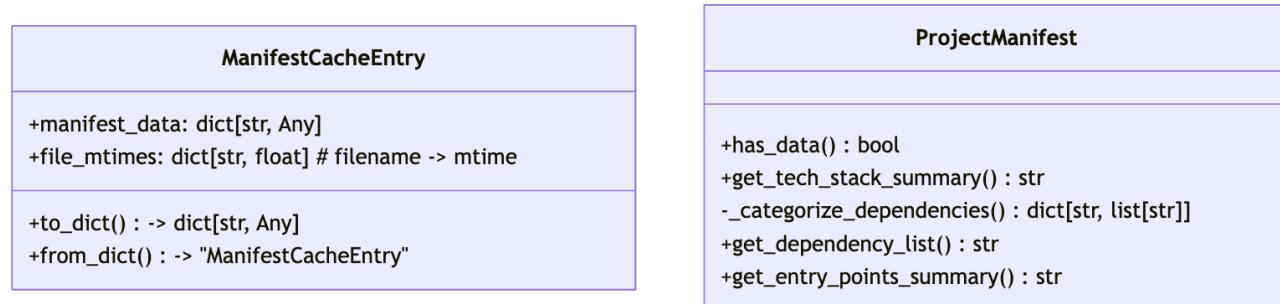
traverse

```
def traverse(path: Path, prefix: str, depth: int) -> None
```

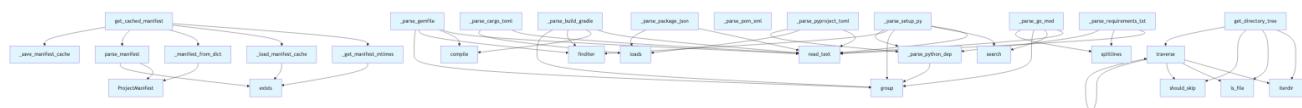
Parameter	Type	Default	Description
path	Path	-	-
prefix	str	-	-
depth	int	-	-

Returns: None

Class Diagram



Call Graph



Relevant Source Files

- src/local_deepwiki/generators/manifest.py:33–52

See Also

- [test_manifest](#) - uses this
- [diagrams](#) - shares 4 dependencies
- [vectorstore](#) - shares 4 dependencies

Wiki Generator Module

Overview

The `wiki.py` module provides the core functionality for generating documentation wikis from indexed code repositories. It contains the `WikiGenerator` class that orchestrates the creation of comprehensive documentation including overview pages, architecture diagrams, module documentation, and file-level documentation.

Classes

WikiGenerator

The `WikiGenerator` class is the `main` component responsible for generating wiki documentation from a vector store of indexed code.

Constructor

```
def __init__(  
    self,  
    wiki_path: Path,  
    vector_store: VectorStore,  
    config: Config | None = None,  
    llm_provider_name: str | None = None,  
):
```

Parameters: - `wiki_path` : Path to the wiki output directory where documentation will be generated - `vector_store` : `VectorStore` instance containing the indexed code repository - `config` : Optional `Config` instance for customization (defaults to global config if not provided) - `llm_provider_name` : Optional override for the LLM provider ("ollama", "anthropic", "openai")

Key Methods

The WikiGenerator class includes several methods for managing the documentation generation process:

- **Status Management:** Methods for loading and saving wiki generation status to track which pages need regeneration
- **Content Generation:** Methods for generating different types of documentation pages including overviews, architecture documentation, module docs, and file-level documentation
- **Page Management:** Methods for writing pages to disk and managing the wiki structure

Functions

generate_wiki

```
async def generate_wiki(
    repo_path: Path,
    wiki_path: Path,
    vector_store: VectorStore,
    index_status: IndexStatus,
    config: Config | None = None,
    llm_provider: str | None = None,
    progress_callback: ProgressCallback | None = None,
    full_rebuild: bool = False,
) -> WikiStructure:
```

A convenience function that provides a high-level interface for generating wiki documentation.

Parameters: - `repo_path` : Path to the source repository - `wiki_path` : Path where the wiki documentation should be generated - `vector_store` : [VectorStore](#) instance with indexed repository content - `index_status` : [IndexStatus](#) object tracking the indexing state - `config` : Optional [Config](#) instance for customization - `llm_provider` : Optional LLM provider override - `progress_callback` : Optional callback for tracking generation progress - `full_rebuild` : Boolean flag to force complete regeneration of all pages

Returns: - `WikiStructure` : Object representing the generated wiki structure

Usage Examples

Basic Wiki Generation

```
from pathlib import Path
from local_deepwiki.generators.wiki import generate_wiki

# Generate wiki documentation
wiki_structure = await generate_wiki(
    repo_path=Path("./my-project"),
    wiki_path=Path("./wiki-output"),
    vector_store=my_vector_store,
    index_status=my_index_status
)
```

Custom Configuration

```
from local_deepwiki.generators.wiki import WikiGenerator

# Create generator with custom configuration
generator = WikiGenerator(
    wiki_path=Path("./docs"),
    vector_store=my_vector_store,
    config=my_config,
    llm_provider_name="anthropic"
)

# Generate the wiki
await generator.generate(
    repo_path=Path("./my-project"),
    index_status=my_index_status
)
```

Related Components

The WikiGenerator integrates with several other components from the local_deepwiki system:

- **VectorStore**: Provides indexed code content for documentation generation
- **Config**: Supplies configuration settings for the generation process
- **EntityRegistry**: Manages cross-linking between documentation pages

- **API Documentation Generator:** Generates API documentation for files
- **Call Graph Generator:** Creates call graph information for code analysis
- **Diagram Generators:** Creates various types of diagrams including class diagrams, dependency graphs, and language charts

The module also works with external components for LLM integration through providers like Ollama, Anthropic, and OpenAI to generate intelligent documentation content.

API Reference

`class WikiGenerator`

Generate wiki documentation from indexed code.

Methods:

`__init__`

```
def __init__(wiki_path: Path, vector_store: VectorStore, config: Config | None = None, llm_provider_name: str | None = None)
```

Initialize the wiki generator.

Parameter	Type	Default	Description
<code>wiki_path</code>	<code>Path</code>	-	Path to wiki output directory.
<code>vector_store</code>	<code>VectorStore</code>	-	Vector store with indexed code.
<code>config</code>	<code>Config None</code>	<code>None</code>	Optional configuration.
<code>llm_provider_name</code>	<code>str None</code>	<code>None</code>	Override LLM provider ("ollama", "anthropic", "openai").

`generate`

```
async def generate(index_status: IndexStatus, progress_callback: ProgressCallback | None = None, full_index: bool | None = None)
```

Generate wiki documentation for the indexed repository.

Parameter	Type	Default	Description
index_status	IndexStatus	-	The index status with file information.
progress_callback	ProgressCallback None	None	Optional progress callback.
full_rebuild	bool	False	If True, regenerate all pages. Otherwise, only regenerate changed pages.

Functions

`generate_wiki`

```
async def generate_wiki(repo_path: Path, wiki_path: Path, vector_store: VectorStore, index_status: IndexStatus) -> None:
```

Convenience function to generate wiki documentation.

Parameter	Type	Default	Description
repo_path	Path	-	Path to the repository.
wiki_path	Path	-	Path for wiki output.
vector_store	VectorStore	-	Indexed vector store.
index_status	IndexStatus	-	Index status.
config	Config None	None	Optional configuration.
llm_provider	str None	None	Optional LLM provider override.
progress_callback	ProgressCallback None	None	Optional progress callback.
full_rebuild	bool	False	If True, regenerate all pages. Otherwise, only regenerate changed pages.

Returns: WikiStructure

Class Diagram



Call Graph



Relevant Source Files

- src/local_deepwiki/generators/wiki.py:66–1160

See Also

- [server](#) - uses this

test_api_docs.py

File Overview

This test file contains unit tests for the API documentation generation functionality in the local_deepwiki project. It tests various components involved in extracting and formatting API documentation from Python source code, including docstring parsing, function signature extraction, and class signature extraction.

Classes

TestAPIDocExtractor

Test class for the [APIDocExtractor](#) functionality.

Key Methods: - `extractor()` - Pytest fixture that returns an [APIDocExtractor](#) instance -
- `test_extract_from_file()` - Tests extracting documentation from a Python file by creating a temporary file with sample code and verifying the extraction process

TestGetFileApiDocs

Test class for the convenience function that gets API documentation for files.

Key Methods: - `test_file_with_content()` - Tests getting API docs for a file containing a function with proper docstring formatting, parameters, and return type annotations

Usage Examples

Based on the test structure shown, the components are used as follows:

```
# Creating an extractor instance
extractor = APIDocExtractor()

# Testing file extraction with sample content
source = '''
def process(value: int = 10) -> str:
    """Process a value.

    Args:
        value: The value to process.

    Returns:
        The processed string.
    """
    return str(value)
'''

# Writing to temporary file for testing
test_file = tmp_path / "processor.py"
test_file.write_text(source)
```

Related Components

This test file imports and tests functionality from several modules:

- **CodeParser** from `local_deepwiki.core.parser` - Used for parsing Python source code
- **APIDocExtractor** - Main class for extracting API documentation
- **ClassSignature** and **FunctionSignature** - Data structures for representing code signatures
- **Parameter** - Represents function/method parameters
- Various utility functions including:
 - `extract_class_signature`
 - `extract_function_signature`
 - `extract_python_decorators`
 - `extract_python_docstring`
 - `extract_python_parameters`
 - `extract_python_return_type`
 - `format_function_signature_line`
 - `format_parameter`

- `generate_api_reference_markdown`
- `get_file_api_docs`
- `parse_docstring`
- `parse_google_docstring`

The tests use pytest fixtures and temporary file handling via `tmp_path` to create isolated test environments for validating the API documentation extraction functionality.

API Reference

class `TestParameter`

Test `Parameter` dataclass.

Methods:

`test_basic_parameter`

```
def test_basic_parameter()
```

Test creating a basic parameter.

`test_full_parameter`

```
def test_full_parameter()
```

Test creating a parameter with all fields.

class `TestExtractPythonParameters`

Test Python parameter extraction.

Methods:

`parser`

```
def parser()
```

test_simple_parameters

```
def test_simple_parameters(parser)
```

Test extracting simple parameters without types.

Parameter	Type	Default	Description
parser	-	-	-

test_typed_parameters

```
def test_typed_parameters(parser)
```

Test extracting parameters with type hints.

Parameter	Type	Default	Description
parser	-	-	-

test_default_parameters

```
def test_default_parameters(parser)
```

Test extracting parameters with default values.

Parameter	Type	Default	Description
parser	-	-	-

test_typed_default_parameters

```
def test_typed_default_parameters(parser)
```

Test extracting parameters with types and defaults.

Parameter	Type	Default	Description
parser	-	-	-

test_excludes_self

```
def test_excludes_self(parser)
```

Test that self is excluded from method parameters.

Parameter	Type	Default	Description
parser	-	-	-

test_excludes_cls

```
def test_excludes_cls(parser)
```

Test that cls is excluded from classmethod parameters.

Parameter	Type	Default	Description
parser	-	-	-

class TestExtractPythonReturnType

Test Python return type extraction.

Methods:**parser**

```
def parser()
```

test_simple_return_type

```
def test_simple_return_type(parser)
```

Test extracting a simple return type.

Parameter	Type	Default	Description
parser	-	-	-

test_complex_return_type

```
def test_complex_return_type(parser)
```

Test extracting a complex return type.

Parameter	Type	Default	Description
parser	-	-	-

test_no_return_type

```
def test_no_return_type(parser)
```

Test function with no return type.

Parameter	Type	Default	Description
parser	-	-	-

class TestExtractPythonDocstring

Test Python docstring extraction.

Methods:

parser

```
def parser()
```

test_triple_quote_docstring

```
def test_triple_quote_docstring(parser)
```

Test extracting triple-quoted docstring.

Parameter	Type	Default	Description
parser	-	-	-

test_multiline_docstring

```
def test_multiline_docstring(parser)
```

Test extracting multiline docstring.

Parameter	Type	Default	Description
parser	-	-	-

test_no_docstring

```
def test_no_docstring(parser)
```

Test function with no docstring.

Parameter	Type	Default	Description
parser	-	-	-

class TestParseGoogleDocstring

Test Google-style docstring parsing.

Methods:

test_simple_description

```
def test_simple_description()
```

Test parsing simple description.

test_args_section

```
def test_args_section()
```

Test parsing Args section.

test_args_with_types

```
def test_args_with_types()
```

Test parsing Args with type annotations.

test_returns_section

```
def test_returns_section()
```

Test parsing Returns section.

class TestParseNumpyDocstring

Test NumPy-style docstring parsing.

Methods:

test_simple_description

```
def test_simple_description()
```

Test parsing simple description.

test_parameters_section

```
def test_parameters_section()
```

Test parsing Parameters section.

class TestExtractFunctionSignature

Test function signature extraction.

Methods:

parser

```
def parser()
```

test_simple_function

```
def test_simple_function(parser)
```

Test extracting simple function signature.

Parameter	Type	Default	Description
parser	-	-	-

test_async_function

```
def test_async_function(parser)
```

Test extracting async function signature.

Parameter	Type	Default	Description
parser	-	-	-

class TestExtractClassSignature

Test class signature extraction.

Methods:

parser

```
def parser()
```

test_simple_class

```
def test_simple_class(parser)
```

Test extracting simple class signature.

Parameter	Type	Default	Description
parser	-	-	-

test_class_with_inheritance

```
def test_class_with_inheritance(parser)
```

Test extracting class with base classes.

Parameter	Type	Default	Description
parser	-	-	-

class TestFormatParameter

Test parameter formatting.

Methods:**test_simple_param**

```
def test_simple_param()
```

Test formatting simple parameter.

test_typed_param

```
def test_typed_param()
```

Test formatting typed parameter.

test_default_param

```
def test_default_param()
```

Test formatting parameter with default.

test_full_param

```
def test_full_param()
```

Test formatting parameter with type and default.

class TestFormatFunctionSignatureLine

Test function signature line formatting.

Methods:

test_simple_function

```
def test_simple_function()
```

Test formatting simple function.

test_function_with_params

```
def test_function_with_params()
```

Test formatting function with parameters.

test_function_with_return_type

```
def test_function_with_return_type()
```

Test formatting function with return type.

test_async_function

```
def test_async_function()
```

Test formatting async function.

class TestGenerateApiReferenceMarkdown

Test API reference markdown generation.

Methods:

test_empty_input

```
def test_empty_input()
```

Test with no functions or classes.

test_function_documentation

```
def test_function_documentation()
```

Test generating function documentation.

test_class_documentation

```
def test_class_documentation()
```

Test generating class documentation.

test_filters_private_items

```
def test_filters_private_items()
```

Test that private items are filtered by default.

test_includes_private_when_requested

```
def test_includes_private_when_requested()
```

Test including private items when specified.

class TestAPIDocExtractor

Test [APIDocExtractor](#) class.

Methods:**extractor**

```
def extractor()
```

test_extract_from_file

```
def test_extract_from_file(tmp_path, extractor)
```

Test extracting docs from a Python file.

Parameter	Type	Default	Description
tmp_path	-	-	-
extractor	-	-	-

test_extract_unsupported_file

```
def test_extract_unsupported_file(tmp_path, extractor)
```

Test extracting from unsupported file type.

Parameter	Type	Default	Description
tmp_path	-	-	-
extractor	-	-	-

class TestGetFileApiDocs

Test the convenience function.

Methods:

test_file_with_content

```
def test_file_with_content(tmp_path)
```

Test getting API docs for a file with content.

Parameter	Type	Default	Description
tmp_path	-	-	-

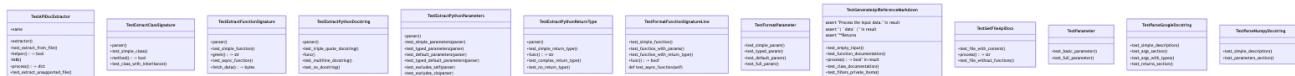
test_file_without_functions

```
def test_file_without_functions(tmp_path)
```

Test getting API docs for file without functions.

Parameter	Type	Default	Description
tmp_path	-	-	-

Class Diagram



Call Graph



Relevant Source Files

- [tests/test_api_docs.py:31-53](#)

See Also

- [api_docs](#) - dependency
- [test_parser](#) - shares 4 dependencies

test_crosslinks.py

File Overview

This file contains comprehensive unit tests for the cross-linking functionality in the wiki documentation system. It tests the ability to automatically create links between wiki pages when entity names (like class names) are mentioned in the content.

Classes

TestCrossLinker

Test suite for the [CrossLinker](#) class that handles automatic cross-linking between wiki pages.

Key Test Methods:

- `test_adds_links_to_prose` - Verifies that entity names in prose text are converted to markdown links
- `test_does_not_link_in_code_blocks` - Ensures entity names inside code blocks remain unchanged
- `test_does_not_self_link` - Prevents entities from linking to themselves on their own page
- `test_relative_paths` - Tests correct relative path calculation between wiki pages
- `test_links_backticked_entities` - Verifies that backticked entity names get linked
- `test_does_not_link_non_entity_inline_code` - Ensures non-entity inline code remains unchanged
- `test_links_qualified_names` - Tests linking of qualified names like `module.ClassName`
- `test_links_simple_qualified_names` - Tests linking of simple qualified names like `generators.WikiGenerator`
- `test_preserves_existing_links` - Ensures existing markdown links are not modified
- `test_links_bold_text` - Tests linking of bold entity names
- `test_links_spaced_aliases` - Tests linking of spaced aliases like "[Vector Store](#)"
- `test_links_bold_spaced_aliases` - Tests linking of bold spaced aliases

TestAddCrossLinks

Test suite for the `add_cross_links` function that processes multiple wiki pages.

Key Test Methods:

- `test_processes_all_pages` - Verifies that the function processes all provided wiki pages

Related Components

This test file works with several components from the cross-linking system:

- **CrossLinker** - The `main` class responsible for adding cross-links to wiki pages
- **EntityRegistry** - Manages registration and lookup of entities that can be cross-linked
- **WikiPage** - Represents individual wiki pages with content to be processed
- **ChunkType** - Enumeration of different code entity types (CLASS, etc.)
- **add_cross_links** - Function that processes multiple pages for cross-linking
- **camel_to_spaced** - Utility function for converting camelCase names to spaced format

Usage Examples

The tests demonstrate typical usage patterns for the cross-linking functionality:

```

# Setting up entity registry and cross-linker
registry = EntityRegistry()
registry.register_entity(
    name="VectorStore",
    entity_type=ChunkType.CLASS,
    wiki_path="files/vectorstore.md",
    file_path="vectorstore.py",
)

linker = CrossLinker(registry)

# Creating a wiki page to process
page = WikiPage(
    path="files/indexer.md",
    title="Indexer",
    content="The indexer uses VectorStore to store embeddings.",
    generated_at=0,
)

# Adding cross-links
result = linker.add_links(page)

```

The tests verify various scenarios including prose text linking, code block preservation, self-link prevention, and different entity name formats (plain, backticked, bold, qualified names, and spaced aliases).

API Reference

class TestCamelToSpaced

Tests for [camel_to_spaced](#) function.

Methods:

test_simple_camel_case

```
def test_simple_camel_case()
```

Test simple CamelCase conversion.

test_multi_word

```
def test_multi_word()
```

Test multi-word CamelCase.

test_acronyms

```
def test_acronyms()
```

Test CamelCase with acronyms.

test_returns_none_for_invalid

```
def test_returns_none_for_invalid()
```

Test that None is returned for non-CamelCase names.

test_already_single_word

```
def test_already_single_word()
```

Test single capitalized word returns None.

class TestEntityRegistry

Tests for [EntityRegistry](#) class.

Methods:

test_register_entity

```
def test_register_entity()
```

Test registering an entity.

test_skips_short_names

```
def test_skips_short_names()
```

Test that short names are not registered.

test_skips_private_names

```
def test_skips_private_names()
```

Test that private names are not registered.

test_skips_excluded_names

```
def test_skips_excluded_names()
```

Test that excluded common names are not registered.

test_register_from_chunks

```
def test_register_from_chunks()
```

Test registering entities from code chunks.

test_get_page_entities

```
def test_get_page_entities()
```

Test getting entities defined in a page.

test_registers_camelcase_aliases

```
def test_registers_camelcase_aliases()
```

Test that CamelCase names get spaced aliases registered.

test_alias_lookup

```
def test_alias_lookup()
```

Test looking up entities by alias.

class TestCrossLinker

Tests for [CrossLinker](#) class.

Methods:

test_adds_links_to_prose

```
def test_adds_links_to_prose()
```

Test that links are added to prose text.

test_does_not_link_in_code_blocks

```
def test_does_not_link_in_code_blocks()
```

Test that links are not added inside code blocks.

test_does_not_self_link

```
def test_does_not_self_link()
```

Test that entities are not linked on their own page.

test_relative_paths

```
def test_relative_paths()
```

Test relative path calculation between pages.

test_links_backticked_entities

```
def test_links_backticked_entities()
```

Test that backticked entity names get linked.

test_does_not_link_non_entity_inline_code

```
def test_does_not_link_non_entity_inline_code()
```

Test that non-entity inline code is preserved unchanged.

test_links_qualified_names

```
def test_links_qualified_names()
```

Test that qualified names like module.ClassName get linked.

test_links_simple_qualified_names

```
def test_links_simple_qualified_names()
```

Test that simple qualified names like module.Class get linked.

test_preserves_existing_links

```
def test_preserves_existing_links()
```

Test that existing markdown links are preserved.

test_links_bold_text

```
def test_links_bold_text()
```

Test that bold entity names get linked.

test_links_spaced_aliases

```
def test_links_spaced_aliases()
```

Test that spaced aliases like '[Vector Store](#)' get linked.

test_links_bold_spaced_aliases

```
def test_links_bold_spaced_aliases()
```

Test that bold spaced aliases get linked.

class TestAddCrossLinks

Tests for [add_cross_links](#) function.

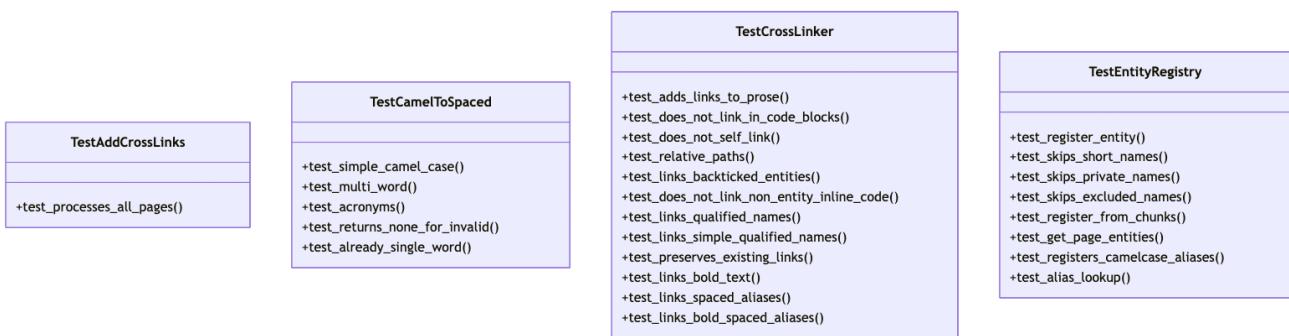
Methods:

`test_processes_all_pages`

```
def test_processes_all_pages()
```

Test that all pages are processed.

Class Diagram



Call Graph



Relevant Source Files

- tests/test_crosslinks.py:12–42

See Also

- [crosslinks](#) - dependency

test_indexer.py

File Overview

This test module validates the functionality of the repository indexer component, focusing on batched processing capabilities and schema migration features. The tests ensure that the indexer can handle large datasets efficiently through batching and maintains backward compatibility through proper schema versioning.

Classes

TestBatchedProcessing

Tests the batched processing functionality of the repository indexer to ensure efficient handling of large code repositories.

Key Test Methods: - `test_processes_chunks_in_batches` - Validates that code chunks are processed in configurable batch sizes - `test_incremental_update_with_batching` - Tests incremental updates using batched operations - `test_empty_batch_handling` - Ensures proper handling of empty batches

TestSchemaMigration

Tests the schema migration system that handles version upgrades of the index status format.

Key Test Methods: - `test_current_schema_version_exists` - Verifies the current schema version constant is defined - `test_needs_migration_old_version` - Tests detection of outdated schema versions requiring migration - `test_needs_migration_current_version` - Validates that current schema versions don't trigger migration - `test_migrate_status_updates_version` - Tests that migration properly updates schema version - `test_migrate_status_preserves_data` - Ensures data integrity during migration - `test_load_status_handles_legacy_files` - Tests loading of legacy status files - `test_save_status_includes_schema_version` - Validates schema version is saved with status - `test_migration_triggered_on_load` - Tests automatic migration triggering during status loading

TestChunkingConfigBatchSize

Tests configuration of batch sizes for chunking operations (class definition not shown in provided code).

TestBatchSizeConfiguration

Tests batch size configuration settings (class definition not shown in provided code).

Functions

test_index_status_model_default_schema_version

```
async def test_index_status_model_default_schema_version(self):
```

Tests that the IndexStatus model defaults to `schema_version=1` when created without explicitly setting the version.

Test Validation: - Creates an IndexStatus instance with basic required fields - Asserts the default schema_version is set to 1

test_migrate_status_preserves_data

```
def test_migrate_status_preserves_data(self):
```

Validates that schema migration preserves all existing data fields during the upgrade process.

Test Process: - Creates an IndexStatus with sample data including repository path, timestamps, file counts, and language statistics - Performs migration using the `_migrate_status` function - Verifies all original data is preserved in the migrated status

Usage Examples

Testing Schema Migration

```
# Create a status with old schema version
status = IndexStatus(
    repo_path="/test/repo",
    indexed_at=1234567890.0,
    total_files=10,
    total_chunks=100,
    languages={"python": 8, "javascript": 2},
    schema_version=1,
)

# Migrate to current schema
migrated, _ = _migrate_status(status)

# Verify data preservation
assert migrated.repo_path == "/test/repo"
assert migrated.total_files == 10
```

Testing Default Schema Version

```
# Create status without explicit schema version
status = IndexStatus(
    repo_path="/test",
    indexed_at=1.0,
    total_files=0,
    total_chunks=0,
)

# Verify default schema version
assert status.schema_version == 1
```

Related Components

This test module works with several core components:

- **RepositoryIndexer** - The [main](#) indexer class being tested
- **IndexStatus** - Model for tracking indexing status and metadata

- **CodeChunk** - Represents individual code chunks in the index
- **ChunkingConfig** and **Config** - Configuration classes for indexing behavior
- **Language** and **ChunkType** - Enums for categorizing code content

The tests also utilize migration helper functions: - `_migrate_status` - Handles schema version upgrades - `_needs_migration` - Determines if migration is required - `CURRENT_SCHEMA_VERSION` - Constant defining the current schema version

API Reference

class `TestChunkingConfigBatchSize`

Tests for batch_size configuration.

Methods:

`test_default_batch_size`

```
def test_default_batch_size()
```

Test that default batch size is 500.

`test_custom_batch_size`

```
def test_custom_batch_size()
```

Test that batch size can be customized.

class `TestBatchedProcessing`

Tests for batched chunk processing in the indexer.

Methods:

`test_processes_chunks_in_batches`

```
async def test_processes_chunks_in_batches(tmp_path)
```

Test that chunks are processed in batches to limit memory usage.

Parameter	Type	Default	Description
tmp_path	-	-	-

mock_create_or_update_table

```
async def mock_create_or_update_table(chunks)
```

Parameter	Type	Default	Description
chunks	-	-	-

mock_add_chunks

```
async def mock_add_chunks(chunks)
```

Parameter	Type	Default	Description
chunks	-	-	-

mock_delete_chunks_by_file

```
async def mock_delete_chunks_by_file(file_path)
```

Parameter	Type	Default	Description
file_path	-	-	-

test_incremental_update_with_batching

```
async def test_incremental_update_with_batching(tmp_path)
```

Test that incremental updates work with batched processing.

Parameter	Type	Default	Description
tmp_path	-	-	-

mock_add_chunks

```
async def mock_add_chunks(chunks)
```

Parameter	Type	Default	Description
chunks	-	-	-

mock_delete_chunks_by_file

```
async def mock_delete_chunks_by_file(file_path)
```

Parameter	Type	Default	Description
file_path	-	-	-

mock_create_or_update_table

```
async def mock_create_or_update_table(chunks)
```

Parameter	Type	Default	Description
chunks	-	-	-

test_empty_batch_handling

```
async def test_empty_batch_handling(tmp_path)
```

Test that empty repositories are handled correctly.

Parameter	Type	Default	Description
tmp_path	-	-	-

class TestBatchSizeConfiguration

Tests for batch size in config.

Methods:**test_batch_size_in_full_config**

```
def test_batch_size_in_full_config()
```

Test that batch size is accessible in full config.

test_batch_size_validation

```
def test_batch_size_validation()
```

Test that batch size accepts positive integers.

class TestSchemaMigration

Tests for schema version migration.

Methods:**test_current_schema_version_exists**

```
def test_current_schema_version_exists()
```

Test that CURRENT_SCHEMA_VERSION is defined.

test_needs_migration_old_version

```
def test_needs_migration_old_version()
```

Test that old schema versions need migration.

test_needs_migration_current_version

```
def test_needs_migration_current_version()
```

Test that current schema version doesn't need migration.

test_migrate_status_updates_version

```
def test_migrate_status_updates_version()
```

Test that migration updates the schema version.

test_migrate_status_preserves_data

```
def test_migrate_status_preserves_data()
```

Test that migration preserves existing data.

test_load_status_handles_legacy_files

```
async def test_load_status_handles_legacy_files(tmp_path)
```

Test that loading status handles legacy files without schema_version.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_save_status_includes_schema_version

```
async def test_save_status_includes_schema_version(tmp_path)
```

Test that saved status includes the current schema version.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_index_status_model_default_schema_version

```
async def test_index_status_model_default_schema_version()
```

Test that IndexStatus defaults to schema_version=1.

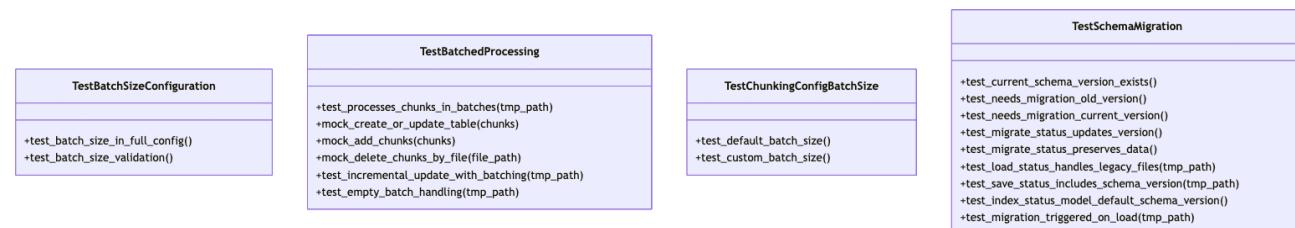
test_migration_triggered_on_load

```
async def test_migration_triggered_on_load(tmp_path)
```

Test that migration is triggered when loading old schema version.

Parameter	Type	Default	Description
tmp_path	-	-	-

Class Diagram



Call Graph



Relevant Source Files

- tests/test_indexer.py:20–31

See Also

- [config](#) - dependency
- [server](#) - shares 5 dependencies

test_manifest.py

File Overview

This test file contains comprehensive unit tests for the manifest parsing and caching functionality in the local_deepwiki project. It tests the ability to parse various project manifest files (pyproject.toml, package.json, requirements.txt, go.mod, Cargo.toml) and includes sophisticated caching mechanisms to improve performance.

Classes

TestProjectManifest

Tests the [ProjectManifest](#) dataclass functionality, focusing on data validation and state checking.

Key Methods: - `test_has_data_empty()` - Verifies that empty manifests correctly report having no data
- `test_has_data_with_name()` - Tests that manifests with names are detected as having data
- `test_has_data_with_dependencies()` - Confirms that manifests with dependencies are recognized as containing data

TestParsePyprojectToml

Tests parsing of Python pyproject.toml files, which contain project metadata, dependencies, and configuration.

Key Methods: - `test_parse_basic_pyproject()` - Tests parsing of a complete pyproject.toml with project metadata, dependencies, optional dependencies, and scripts

TestParseRequirementsTxt

Tests parsing of Python requirements.txt files for dependency management.

Key Methods: - `test_parse_basic_requirements()` - Tests parsing of requirements.txt files with version specifications and comments

TestParseGoMod

Tests parsing of Go module files (go.mod) for Go project dependency management.

Key Methods: - `test_parse_basic_go_mod()` - Tests parsing of go.mod files with module names, Go versions, and dependencies

TestMultipleManifests

Tests handling of projects that contain multiple manifest files and the precedence rules between them.

Key Methods: - `test_pyproject_takes_precedence()` - Verifies that pyproject.toml takes precedence over requirements.txt when both are present

TestManifestCaching

Tests the sophisticated caching system that improves performance by avoiding re-parsing unchanged manifest files.

Key Methods: - `test_get_manifest_mtis_empty_repo()` - Tests modification time retrieval for repositories without manifest files - `test_get_manifest_mtis_with_files()` - Tests modification time retrieval when manifest files are present - `test_cache_entry_serialization()` - Tests serialization and deserialization of cache entries - `test_cache_valid_when_unchanged()` - Verifies cache validity when files haven't changed - `test_cache_invalid_when_file_modified()` - Tests cache invalidation when files are modified - `test_cache_invalid_when_file_added()` - Tests cache invalidation when new manifest files are added - `test_cache_invalid_when_file_removed()` - Tests cache invalidation when manifest files are removed - `test_get_cached_manifest_creates_cache()` - Tests cache file creation on first use - `test_get_cached_manifest_uses_cache()` - Tests cache utilization on subsequent calls - `test_get_cached_manifest_invalidates_on_change()` - Tests cache invalidation and re-parsing when files change - `test_get_cached_manifest_default_cache_dir()` - Tests default cache directory usage

Functions Tested

The test file validates several functions from the manifest module:

```
_get_manifest_mtimes(root)
```

Returns modification times for manifest files in a directory.

```
_is_cache_valid(entry, new_mtimes)
```

Determines if a cache entry is still valid by comparing file modification times.

```
get_cached_manifest(root, cache_dir=None)
```

Retrieves a parsed manifest, using cache when possible or creating/updating cache as needed.

```
parse_manifest(path)
```

Parses manifest files in a directory and returns a [ProjectManifest](#) object.

Usage Examples

Testing Cache Functionality

```
# Create a project with manifest
root = Path("/path/to/project")
(root / "pyproject.toml").write_text("""
[project]
name = "my-project"
dependencies = ["requests"]
""")

# Get cached manifest (creates cache on first call)
manifest = get_cached_manifest(root)
assert manifest.name == "my-project"

# Subsequent calls use cache
cached_manifest = get_cached_manifest(root)
```

Testing Multiple Manifest Precedence

```
# Create directory with both pyproject.toml and requirements.txt
root = Path("/path/to/project")
(root / "pyproject.toml").write_text("""
[project]
name = "from-pyproject"
dependencies = ["flask"]
""")
(root / "requirements.txt").write_text("requests")

# pyproject.toml takes precedence
manifest = parse_manifest(root)
assert manifest.name == "from-pyproject"
```

Related Components

This test file works with several components from the local_deepwiki.generators.manifest module:

- [ManifestCacheEntry](#) - Handles cache entry serialization and validation
- [ProjectManifest](#) - Core data structure representing parsed project metadata
- [get_directory_tree](#) - Function for directory structure analysis (imported but not shown in test methods)

The caching system uses JSON serialization and file modification time tracking to provide efficient manifest parsing across multiple tool invocations.

API Reference

`class TestProjectManifest`

Tests for [ProjectManifest](#) dataclass.

Methods:

`test_has_data_empty`

```
def test_has_data_empty()
```

Empty manifest has no data.

test_has_data_with_name

```
def test_has_data_with_name()
```

Manifest with name has data.

test_has_data_with_dependencies

```
def test_has_data_with_dependencies()
```

Manifest with dependencies has data.

test_get_tech_stack_summary_empty

```
def test_get_tech_stack_summary_empty()
```

Empty manifest returns default message.

test_get_tech_stack_summary_with_language

```
def test_get_tech_stack_summary_with_language()
```

Manifest with language shows it in summary.

test_get_dependency_list_formatted

```
def test_get_dependency_list_formatted()
```

Dependencies are formatted correctly.

class TestParsePyprojectToml

Tests for parsing pyproject.toml files.

Methods:

test_parse_basic_pyproject

```
def test_parse_basic_pyproject()
```

Parse a basic pyproject.toml.

class TestParsePackageJson

Tests for parsing package.json files.

Methods:

test_parse_basic_package_json

```
def test_parse_basic_package_json()
```

Parse a basic package.json.

class TestParseRequirementsTxt

Tests for parsing requirements.txt files.

Methods:

test_parse_basic_requirements

```
def test_parse_basic_requirements()
```

Parse a basic requirements.txt.

class TestParseCargoToml

Tests for parsing Cargo.toml files.

Methods:

test_parse_basic_cargo

```
def test_parse_basic_cargo()
```

Parse a basic Cargo.toml.

class TestParseGoMod

Tests for parsing go.mod files.

Methods:

test_parse_basic_go_mod

```
def test_parse_basic_go_mod()
```

Parse a basic go.mod.

class TestGetDirectoryTree

Tests for directory tree generation.

Methods:

test_basic_tree

```
def test_basic_tree()
```

Generate a basic directory tree.

test_skips_hidden_dirs

```
def test_skips_hidden_dirs()
```

Hidden directories are skipped.

test_skips_node_modules

```
def test_skips_node_modules()
```

node_modules is skipped.

test_respects_max_items

```
def test_respects_max_items()
```

Respects max_items limit.

class TestMultipleManifests

Tests for handling multiple manifest files.

Methods:

test_pyproject_takes_precedence

```
def test_pyproject_takes_precedence()
```

pyproject.toml takes precedence over requirements.txt.

class TestManifestCaching

Tests for manifest caching functionality.

Methods:

test_get_manifest_mtimes_empty_repo

```
def test_get_manifest_mtimes_empty_repo()
```

Empty repo returns no mtimes.

test_get_manifest_mtimes_with_files

```
def test_get_manifest_mtimes_with_files()
```

Returns mtimes for existing manifest files.

test_cache_entry_serialization

```
def test_cache_entry_serialization()
```

Cache entry can be serialized and deserialized.

test_cache_valid_when_unchanged

```
def test_cache_valid_when_unchanged()
```

Cache is valid when files haven't changed.

test_cache_invalid_when_file_modified

```
def test_cache_invalid_when_file_modified()
```

Cache is invalid when a file is modified.

test_cache_invalid_when_file_added

```
def test_cache_invalid_when_file_added()
```

Cache is invalid when a new manifest file is added.

test_cache_invalid_when_file_removed

```
def test_cache_invalid_when_file_removed()
```

Cache is invalid when a manifest file is removed.

test_get_cached_manifest_creates_cache

```
def test_get_cached_manifest_creates_cache()
```

[get_cached_manifest](#) creates cache file on first call.

test_get_cached_manifest_uses_cache

```
def test_get_cached_manifest_uses_cache()
```

[get_cached_manifest](#) uses cache on subsequent calls.

test_get_cached_manifest_invalidates_on_change

```
def test_get_cached_manifest_invalidates_on_change()
```

[get_cached_manifest](#) re-parses when file changes.

test_get_cached_manifest_default_cache_dir

```
def test_get_cached_manifest_default_cache_dir()
```

[get_cached_manifest](#) uses .deepwki in repo by default.

Class Diagram



Call Graph



Relevant Source Files

- [tests/test_manifest.py:19–61](#)

See Also

- [manifest](#) - dependency
- [test_indexer](#) - shares 3 dependencies

test_models.py

File Overview

This file contains unit tests for the string representation (`__repr__`) methods of model classes in the `local_deepwiki` package. The tests verify that model objects display meaningful and properly formatted string representations.

Classes

TestModelRepr

A test class that contains methods to verify the string representation of various model objects from the `local_deepwiki.models` module.

Methods: - `test_code_chunk_repr_with_name` - Tests `CodeChunk` repr for named chunks -
`test_code_chunk_repr_without_name` - Tests `CodeChunk` repr for unnamed chunks
- `test_file_info_repr` - Tests `FileInfo` repr - `test_file_info_repr_no_language` - Tests `FileInfo` repr without language - `test_index_status_repr` - Tests `IndexStatus` repr - `test_wiki_page_repr` - Tests `WikiPage` repr - `test_wiki_structure_repr` - Tests `WikiStructure` repr -
`test_search_result_repr` - Tests `SearchResult` repr - `test_search_result_repr_no_name` - Tests `SearchResult` repr without name - `test_wiki_page_status_repr` - Tests `WikiPageStatus` repr -
`test_wiki_generation_status_repr` - Tests `WikiGenerationStatus` repr

TestCodeChunkToVectorRecord

A test class for testing conversion of `CodeChunk` objects to vector records (implementation not shown in provided code).

Test Methods

test_code_chunk_repr_with_name

Tests the string representation of a CodeChunk object that has a name.

```
def test_code_chunk_repr_with_name(self):
    chunk = CodeChunk(
        id="test_id",
        file_path="src/main.py",
        language=Language.PYTHON,
        chunk_type=ChunkType.FUNCTION,
        name="my_function",
        content="def my_function(): pass",
        start_line=10,
        end_line=15,
    )
    result = repr(chunk)
    # Verifies result contains "CodeChunk", "function", "my_function", "src/main.py:10-15"
```

test_code_chunk_repr_without_name

Tests the string representation of a CodeChunk object without a name.

```
def test_code_chunk_repr_without_name(self):
    chunk = CodeChunk(
        id="test_id",
        file_path="src/module.py",
        language=Language.PYTHON,
        chunk_type=ChunkType.MODULE,
        content="# module",
        start_line=1,
        end_line=5,
    )
    result = repr(chunk)
    # Verifies result contains "CodeChunk", "module", "src/module.py:1-5"
```

test_file_info_repr

Tests the string representation of a FileInfo object.

```
def test_file_info_repr(self):
    info = FileInfo(
        path="src/utils.py",
        language=Language.PYTHON,
        size_bytes=1024,
        last_modified=1234567890.0,
        hash="abc123",
        chunk_count=5,
    )
    result = repr(info)
    # Verifies result contains "FileInfo", "src/utils.py", "python", "5 chunks"
```

test_index_status_repr

Tests the string representation of an IndexStatus object.

```
def test_index_status_repr(self):
    status = IndexStatus(
        repo_path="/home/user/project",
        indexed_at=1234567890.0,
        total_files=25,
        total_chunks=150,
    )
    result = repr(status)
    # Verifies result contains "IndexStatus", "/home/user/project", "25 files", "150 chunks"
```

test_wiki_page_repr

Tests the string representation of a WikiPage object.

```
def test_wiki_page_repr(self):
    page = WikiPage(
        path="modules/core.md",
        title="Core Module",
        content="# Core Module\n\nContent here.",
        generated_at=1234567890.0,
    )
    result = repr(page)
    # Verifies result contains "WikiPage", "modules/core.md", "Core Module"
```

Related Components

This test file works with the following model classes from `local_deepwiki.models` :

- **ChunkType** - Enumeration for code chunk types
- **CodeChunk** - Represents a chunk of code with metadata
- **FileInfo** - Contains file metadata and statistics
- **IndexStatus** - Tracks repository indexing status
- **Language** - Enumeration for programming languages
- **SearchResult** - Represents search results
- **WikiGenerationStatus** - Tracks wiki generation progress
- **WikiPage** - Represents a generated wiki page
- **WikiPageStatus** - Tracks wiki page status
- **WikiStructure** - Represents wiki organization structure

The tests use pytest framework for test execution and assertions.

API Reference

`class TestCodeChunkToVectorRecord`

Tests for `CodeChunk.to_vector_record` method.

Methods:

`test_basic_conversion`

```
def test_basic_conversion()
```

Test basic chunk to vector record conversion.

`test_with_vector`

```
def test_with_vector()
```

Test conversion with vector embedding.

test_with_optional_fields

```
def test_with_optional_fields()
```

Test conversion with optional fields populated.

test_none_fields_become_empty_strings

```
def test_none_fields_become_empty_strings()
```

Test that None fields are converted to empty strings.

test_all_languages

```
def test_all_languages()
```

Test conversion works for all supported languages.

test_all_chunk_types

```
def test_all_chunk_types()
```

Test conversion works for all chunk types.

class TestModelRepr

Tests for model **repr** methods.

Methods:

test_code_chunk_repr_with_name

```
def test_code_chunk_repr_with_name()
```

Test CodeChunk repr with a named chunk.

test_code_chunk_repr_without_name

```
def test_code_chunk_repr_without_name()
```

Test CodeChunk repr without a name.

test_file_info_repr

```
def test_file_info_repr()
```

Test FileInfo repr.

test_file_info_repr_no_language

```
def test_file_info_repr_no_language()
```

Test FileInfo repr with no detected language.

test_index_status_repr

```
def test_index_status_repr()
```

Test IndexStatus repr.

test_wiki_page_repr

```
def test_wiki_page_repr()
```

Test WikiPage repr.

test_wiki_structure_repr

```
def test_wiki_structure_repr()
```

Test WikiStructure repr.

test_search_result_repr

```
def test_search_result_repr()
```

Test SearchResult repr.

test_search_result_repr_no_name

```
def test_search_result_repr_no_name()
```

Test SearchResult repr when chunk has no name.

test_wiki_page_status_repr

```
def test_wiki_page_status_repr()
```

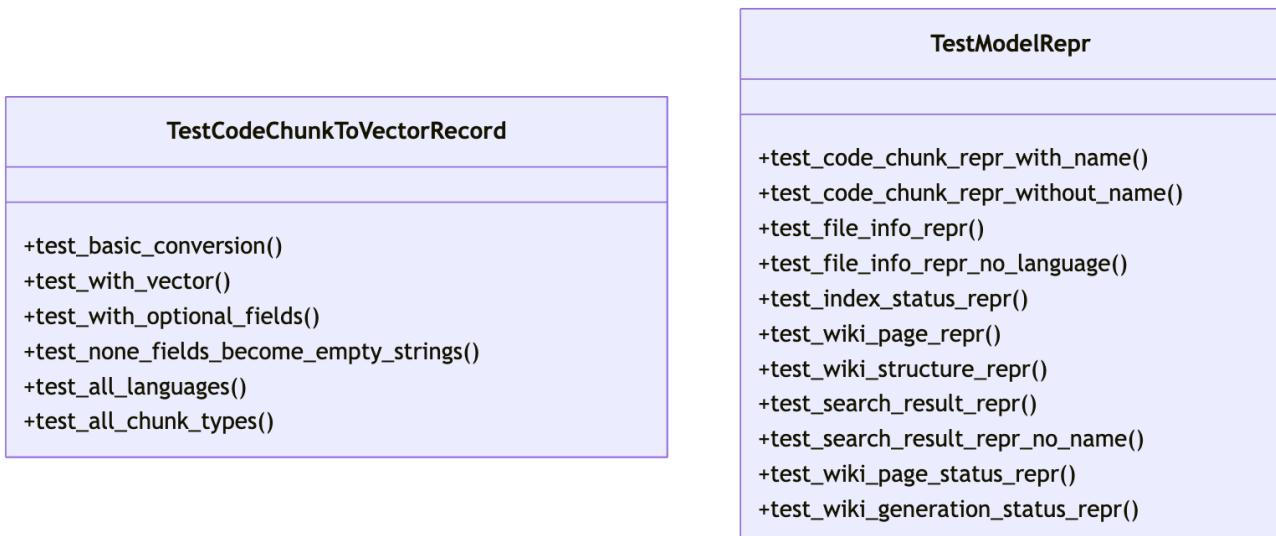
Test WikiPageStatus repr.

test_wiki_generation_status_repr

```
def test_wiki_generation_status_repr()
```

Test WikiGenerationStatus repr.

Class Diagram



Call Graph



Relevant Source Files

- tests/test_models.py:21–144

See Also

- `test_indexer` - shares 3 dependencies

test_parser.py

File Overview

This file contains comprehensive test suites for the code parsing functionality in the local_deepwiki system. It tests the [CodeParser](#) class and various helper functions for parsing source code, extracting metadata, and handling different programming languages.

Classes

TestCodeParser

Test suite for the [CodeParser](#) class that handles source code parsing and language detection.

Key Methods: - `setup_method()` - Initializes a [CodeParser](#) instance for testing -
`test_detect_language_python()` - Tests Python file detection based on file extensions (.py, .pyi) -
`test_detect_language_javascript()` - Tests JavaScript file detection based on file extensions

TestNodeHelpers

Test suite for node helper functions that extract information from parsed syntax tree nodes.

Key Methods: - `setup_method()` - Sets up test fixtures with a [CodeParser](#) instance -
`test_get_node_text()` - Tests extracting source code text from syntax tree nodes -
`test_get_node_name_python_function()` - Tests extracting function names from Python syntax nodes

TestCommentHelpers

Test suite for comment-related helper functions (referenced but implementation not shown in provided code).

TestDocstringExtraction

Comprehensive test suite for extracting documentation strings from various programming languages.

Key Methods:

- `setup_method()` - Initializes parser for docstring extraction tests -
- `test_python_docstring()` - Tests Python docstring extraction from function definitions -
- `test_go_single_line_comment()` - Tests Go single-line comment extraction -
- `test_go_multi_line_comments()` - Tests Go multi-line comment extraction -
- `test_rust_single_line_doc_comment()` - Tests Rust single-line documentation comment extraction -
- `test_rust_multi_line_doc_comments()` - Tests Rust multi-line documentation comment extraction -
- `test_ruby_single_line_comment()` - Tests Ruby single-line comment extraction -
- `test_ruby_multi_line_comments()` - Tests Ruby multi-line comment extraction -
- `test_javascript_jsdoc_block()` - Tests JavaScript JSDoc block extraction -
- `test_java_javadoc_block()` - Tests Java Javadoc block extraction -
- `test_cpp_doxygen_triple_slash()` - Tests C++ Doxygen triple-slash comment extraction -
- `test_no_docstring()` - Tests handling of code without documentation

TestLargeFileHandling

Test suite for handling large source code files (referenced but implementation not shown in provided code).

Functions

The tests verify various imported helper functions:

- `_collect_preceding_comments()` - Collects comments that precede code elements
- `_compute_file_hash()` - Computes hash values for files
- `_read_file_content()` - Reads file content with appropriate handling
- `_strip_line_comment_prefix()` - Removes comment prefixes from lines
- `find_nodes_by_type()` - Finds syntax tree nodes by their type
- `get_docstring()` - Extracts documentation strings from code
- `get_node_name()` - Gets the name identifier from syntax tree nodes
- `get_node_text()` - Extracts source text from syntax tree nodes

Usage Examples

Testing Language Detection

```
parser = CodeParser()
assert parser.detect_language(Path("test.py")) == Language.PYTHON
assert parser.detect_language(Path("test.js")) == Language.JAVASCRIPT
```

Testing Docstring Extraction

```
code = b'''def hello():
    """This is a docstring."""
    pass'''
root = parser.parse_source(code, Language.PYTHON)
func_node = root.children[0]
docstring = get_docstring(func_node, code, Language.PYTHON)
assert docstring == "This is a docstring."
```

Testing Node Text Extraction

```
code = b"def foo(): pass"
root = parser.parse_source(code, Language.PYTHON)
func_node = root.children[0]
text = get_node_text(func_node, code)
assert text == "def foo(): pass"
```

Related Components

This test file works with several core components:

- **CodeParser** - The [main](#) parser class being tested
- **Language** - Enumeration of supported programming languages
- Various helper functions from the `local_deepwiki.core.parser` module
- Constants like `HASH_CHUNK_SIZE` and `MMAP_THRESHOLD_BYTES` for file processing configuration

The tests use `pytest` fixtures and temporary files for isolated testing of parsing functionality across multiple programming languages.

API Reference

class TestCodeParser

Test suite for [CodeParser](#).

Methods:

setup_method

```
def setup_method()
```

Set up test fixtures.

test_detect_language_python

```
def test_detect_language_python()
```

Test Python language detection.

test_detect_language_javascript

```
def test_detect_language_javascript()
```

Test JavaScript language detection.

test_detect_language_typescript

```
def test_detect_language_typescript()
```

Test TypeScript language detection.

test_detect_language_go

```
def test_detect_language_go()
```

Test Go language detection.

test_detect_language_rust

```
def test_detect_language_rust()
```

Test Rust language detection.

test_detect_language_unsupported

```
def test_detect_language_unsupported()
```

Test unsupported file extensions.

test_parse_python_file

```
def test_parse_python_file(tmp_path)
```

Test parsing a Python file.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_parse_javascript_file

```
def test_parse_javascript_file(tmp_path)
```

Test parsing a JavaScript file.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_parse_source_string

```
def test_parse_source_string()
```

Test parsing source code from a string.

test_get_file_info

```
def test_get_file_info(tmp_path)
```

Test getting file info.

Parameter	Type	Default	Description
tmp_path	-	-	-

class TestNodeHelpers

Test node helper functions.

Methods:**setup_method**

```
def setup_method()
```

Set up test fixtures.

test_get_node_text

```
def test_get_node_text()
```

Test extracting text from nodes.

test_get_node_name_python_function

```
def test_get_node_name_python_function()
```

Test getting name from Python function.

test_get_node_name_python_class

```
def test_get_node_name_python_class()
```

Test getting name from Python class.

class TestCommentHelpers

Tests for comment collection helper functions.

Methods:

test_strip_line_comment_prefix_single_line

```
def test_strip_line_comment_prefix_single_line()
```

Test stripping prefix from single comment.

test_strip_line_comment_prefix_multi_line

```
def test_strip_line_comment_prefix_multi_line()
```

Test stripping prefix from multiple comments.

test_strip_line_comment_prefix_with_space

```
def test_strip_line_comment_prefix_with_space()
```

Test stripping prefix preserves content after space.

test_strip_line_comment_prefix_no_space

```
def test_strip_line_comment_prefix_no_space()
```

Test stripping prefix without space after prefix.

class TestDocstringExtraction

Tests for docstring extraction from various languages.

Methods:

setup_method

```
def setup_method()
```

Set up test fixtures.

test_python_docstring

```
def test_python_docstring()
```

Test extracting Python docstring.

test_go_single_line_comment

```
def test_go_single_line_comment()
```

Test Go single-line doc comment.

test_go_multi_line_comments

```
def test_go_multi_line_comments()
```

Test Go multi-line doc comments.

test_rust_single_line_doc_comment

```
def test_rust_single_line_doc_comment()
```

Test Rust single-line doc comment.

test_rust_multi_line_doc_comments

```
def test_rust_multi_line_doc_comments()
```

Test Rust multi-line doc comments.

test_ruby_single_line_comment

```
def test_ruby_single_line_comment()
```

Test Ruby single-line doc comment.

test_ruby_multi_line_comments

```
def test_ruby_multi_line_comments()
```

Test Ruby multi-line doc comments.

test_javascript_jsdoc_block

```
def test_javascript_jsdoc_block()
```

Test JavaScript JSDoc block comment.

test_java_javadoc_block

```
def test_java_javadoc_block()
```

Test Java Javadoc block comment.

test_cpp_doxygen_triple_slash

```
def test_cpp_doxygen_triple_slash()
```

Test C++ Doxygen triple-slash comments.

test_no_docstring

```
def test_no_docstring()
```

Test function without docstring.

class TestLargeFileHandling

Tests for memory-efficient large file handling.

Methods:

test_mmap_threshold_constant

```
def test_mmap_threshold_constant()
```

Test that MMAP threshold is set to 1 MB.

test_hash_chunk_size_constant

```
def test_hash_chunk_size_constant()
```

Test that hash chunk size is set to 64 KB.

test_read_small_file_directly

```
def test_read_small_file_directly()
```

Test that small files are read directly.

test_read_file_content_preserves_bytes

```
def test_read_file_content_preserves_bytes()
```

Test that file content is preserved exactly.

test_compute_hash_small_file

```
def test_compute_hash_small_file()
```

Test hash computation for small file.

test_compute_hash_empty_file

```
def test_compute_hash_empty_file()
```

Test hash computation for empty file.

test_parser_handles_large_file

```
def test_parser_handles_large_file()
```

Test that parser can handle files above mmap threshold.

test_get_file_info_large_file

```
def test_get_file_info_large_file()
```

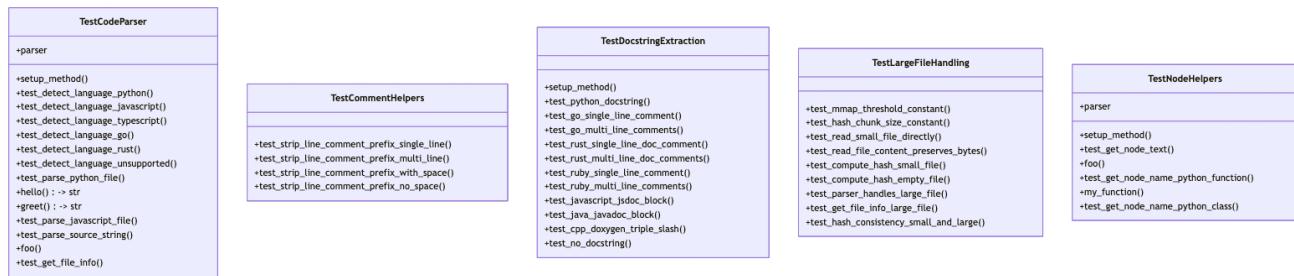
Test get_file_info uses chunked hashing for large files.

test_hash_consistency_small_and_large

```
def test_hash_consistency_small_and_large()
```

Test that hash is consistent regardless of file size.

Class Diagram



Call Graph



Relevant Source Files

- tests/test_parser.py:24–123

See Also

- [test_api_docs](#) - shares 4 dependencies

- [test_indexer](#) - shares 4 dependencies

test_pdf_export.py

File Overview

This test file contains comprehensive unit tests for the PDF export functionality in the local_deepwiki package. It tests various components including the [PdfExporter](#) class, mermaid diagram handling, markdown rendering for PDF output, and the [export_to_pdf](#) convenience function.

Test Classes

TestRenderMarkdownForPdf

Tests PDF-specific markdown rendering functionality.

Methods: - `test_basic_markdown()` - Verifies basic markdown to HTML conversion -
`test_code_blocks()` - Tests fenced code block rendering - `test_table()` - Tests table rendering
(method signature visible but implementation truncated)

TestPdfExporter

Main test class for the [PdfExporter](#) functionality.

Fixtures: - `sample_wiki(tmp_path)` - Creates a sample wiki structure for testing

Methods: - `test_collect_pages_in_order()` - Verifies pages are collected according to TOC order -
`test_extract_paths_from_toc()` - Tests TOC path extraction (method visible but implementation
not shown) - `test_build_toc_html()` - Tests HTML generation for table of contents -
`test_export_singleCreatesPdf()` - Verifies single PDF export functionality -
`test_export_singleWithDirectoryOutput()` - Tests export when output is a directory -
`test_export_separateCreatesMultiplePdfs()` - Tests creation of multiple PDF files -
`test_export_separatePreservesDirectoryStructure()` - Verifies directory structure preservation

TestExportToPdf

Tests for the [export_to_pdf](#) convenience function.

Fixtures: - `simple_wiki(tmp_path)` - Creates a minimal wiki structure for testing

Methods: - `test_raises_for_nonexistent_wiki()` - Tests error handling for invalid wiki paths
(implementation truncated)

TestMermaidHandling

Tests mermaid diagram handling when CLI is not available.

Methods: - `test_mermaid_replaced_with_note()` - Verifies mermaid diagrams are replaced with informational notes - `test_regular_code_blocks_preserved()` - Ensures regular code blocks remain unchanged (implementation truncated)

TestIsMmdcAvailable

Tests the mermaid CLI availability detection.

Methods: - `test_mmdc_available()` - Tests when mmdc CLI tool is available - `test_mmdc_not_av()`
- Tests when mmdc CLI tool is not available (implementation truncated)

TestRenderMermaidToPng

Tests PNG rendering functionality for mermaid diagrams.

Methods: - `test_returns_none_when_mmdc_unavailable()` - Verifies None return when mmdc is unavailable - `test_returns_none_on_cli_error()` - Tests error handling for CLI failures
(implementation truncated)

TestMermaidCliRendering

Tests mermaid rendering when CLI is available.

Methods: - `test_renderers_mermaid_when_cli_available()` - Tests PNG rendering when mmdc CLI is present

Imported Components

The test file imports and tests the following components from `local_deepwiki.export.pdf` :

- `PdfExporter` - Main class for PDF export operations
- `export_to_pdf` - Convenience function for PDF export
- `extract_mermaid_blocks` - Function to extract mermaid diagram blocks
- `extract_title` - Function to extract document titles
- `is_mmdc_available` - Function to check mermaid CLI availability
- `render_markdown_for_pdf` - Function for PDF-specific markdown rendering
- `render_mermaid_to_png` - Function to render mermaid diagrams as PNG
- `render_mermaid_to_svg` - Function to render mermaid diagrams as SVG
- `PRINT_CSS` - CSS constants for PDF styling

Usage Examples

Testing PDF Export

```
# Create a PdfExporter instance
exporter = PdfExporter(sample_wiki, output_path)

# Export as single PDF
result = exporter.export_single()

# Export as separate PDFs
results = exporter.export_separate()
```

Testing Markdown Rendering

```
# Render markdown for PDF
html = render_markdown_for_pdf(markdown_content)

# Render with mermaid handling disabled
html = render_markdown_for_pdf(markdown_content, render_mermaid=False)
```

Testing Mermaid CLI Availability

```
# Check if mermaid CLI is available
available = is_mmdc_available()

# Render mermaid to PNG (returns None if unavailable)
png_data = render_mermaid_to_png("graph TD\nA-->B")
```

The tests use mocking extensively to isolate functionality and avoid dependencies on external tools like WeasyPrint and the mermaid CLI during testing.

API Reference

class `TestRenderMarkdownForPdf`

Tests for PDF-specific markdown rendering.

Methods:

`test_basic_markdown`

```
def test_basic_markdown()
```

Test basic markdown conversion.

`test_code_blocks`

```
def test_code_blocks()
```

Test fenced code blocks.

`test_tables`

```
def test_tables()
```

Test markdown tables.

test_mermaid_blocks_replaced_with_note

```
def test_mermaid_blocks_replaced_with_note()
```

Test that mermaid blocks are replaced with a note when CLI unavailable.

test_multiple_mermaid_blocks

```
def test_multiple_mermaid_blocks()
```

Test handling multiple mermaid blocks when CLI unavailable.

class TestExtractTitle

Tests for title extraction.

Methods:

test_h1_title

```
def test_h1_title(tmp_path: Path)
```

Test extracting H1 title.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_bold_title

```
def test_bold_title(tmp_path: Path)
```

Test extracting bold title.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_fallback_to_filename

```
def test_fallback_to_filename(tmp_path: Path)
```

Test fallback to filename when no title found.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_h1_with_leading_whitespace

```
def test_h1_with_leading_whitespace(tmp_path: Path)
```

Test extracting H1 title with leading whitespace.

Parameter	Type	Default	Description
tmp_path	Path	-	-

class TestPdfExporter

Tests for [PdfExporter](#) class.

Methods:**sample_wiki**

```
def sample_wiki(tmp_path: Path) -> Path
```

Create a sample wiki structure for testing.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_collect_pages_in_order

```
def test_collect_pages_in_order(sample_wiki: Path, tmp_path: Path)
```

Test that pages are collected in TOC order.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_extract_paths_from_toc

```
def test_extract_paths_from_toc(sample_wiki: Path, tmp_path: Path)
```

Test extracting paths from nested TOC.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_build_toc_html

```
def test_build_toc_html(sample_wiki: Path, tmp_path: Path)
```

Test building TOC HTML.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_export_singleCreatesPdf

```
def test_export_singleCreatesPdf(mock_html_class, sample_wiki: Path, tmp_path: Path)
```

Test that export_single creates a PDF file.

Parameter	Type	Default	Description
mock_html_class	-	-	-
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_export_singleWithDirectoryOutput

```
def test_export_singleWithDirectoryOutput(mock_html_class, sample_wiki: Path, tmp_path: Path)
```

Test export_single with directory as output path.

Parameter	Type	Default	Description
mock_html_class	-	-	-
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_export_separateCreatesMultiplePdfs

```
def test_export_separateCreatesMultiplePdfs(mock_html_class, sample_wiki: Path, tmp_path: Path)
```

Test that export_separate creates multiple PDF files.

Parameter	Type	Default	Description
mock_html_class	-	-	-
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_export_separate_preserves_directory_structure

```
def test_export_separate_preserves_directory_structure(mock_html_class, sample_wiki: Path, tmp_path: Path)
```

Test that export_separate preserves directory structure.

Parameter	Type	Default	Description
mock_html_class	-	-	-
sample_wiki	Path	-	-
tmp_path	Path	-	-

class TestExportToPdf

Tests for the [export_to_pdf](#) convenience function.

Methods:**simple_wiki**

```
def simple_wiki(tmp_path: Path) -> Path
```

Create a simple wiki for testing.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_raises_for_nonexistent_wiki

```
def test_raises_for_nonexistent_wiki(tmp_path: Path)
```

Test that export raises for nonexistent wiki path.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_default_output_path_single

```
def test_default_output_path_single(mock_html_class, simple_wiki: Path)
```

Test default output path for single file mode.

Parameter	Type	Default	Description
mock_html_class	-	-	-
simple_wiki	Path	-	-

test_default_output_path_separate

```
def test_default_output_path_separate(mock_html_class, simple_wiki: Path)
```

Test default output path for separate file mode.

Parameter	Type	Default	Description
mock_html_class	-	-	-
simple_wiki	Path	-	-

test_custom_output_path

```
def test_custom_output_path(mock_html_class, simple_wiki: Path, tmp_path: Path)
```

Test custom output path.

Parameter	Type	Default	Description
mock_html_class	-	-	-
simple_wiki	Path	-	-
tmp_path	Path	-	-

test_string_paths_accepted

```
def test_string_paths_accepted(mock_html_class, simple_wiki: Path, tmp_path: Path)
```

Test that string paths are accepted.

Parameter	Type	Default	Description
mock_html_class	-	-	-
simple_wiki	Path	-	-
tmp_path	Path	-	-

class TestPrintCss

Tests for print CSS content.

Methods:

test_print_css_has_page_rules

```
def test_print_css_has_page_rules()
```

Test that print CSS has @page rules.

test_print_css_has_page_numbers

```
def test_print_css_has_page_numbers()
```

Test that print CSS includes page numbers.

test_print_css_avoids_page_breaks_in_code

```
def test_print_css_avoids_page_breaks_in_code()
```

Test that print CSS avoids page breaks inside code blocks.

test_print_css_keeps_headings_with_content

```
def test_print_css_keeps_headings_with_content()
```

Test that print CSS keeps headings with following content.

class TestMermaidHandling

Tests for mermaid diagram handling in PDF export without CLI.

Methods:

test_mermaid_replaced_with_note

```
def test_mermaid_replaced_with_note()
```

Test that mermaid diagrams are replaced with a note when CLI unavailable.

test_regular_code_blocks_preserved

```
def test_regular_code_blocks_preserved()
```

Test that regular code blocks are preserved.

test_mixed_code_blocks

```
def test_mixed_code_blocks()
```

Test document with both mermaid and regular code blocks.

class TestExtractMermaidBlocks

Tests for mermaid block extraction.

Methods:

test_extract_single_block

```
def test_extract_single_block()
```

Test extracting a single mermaid block.

test_extract_multiple_blocks

```
def test_extract_multiple_blocks()
```

Test extracting multiple mermaid blocks.

test_no_mermaid_blocks

```
def test_no_mermaid_blocks()
```

Test content with no mermaid blocks.

test_diagram_code_stripped

```
def test_diagram_code_stripped()
```

Test that diagram code is stripped of whitespace.

class TestIsMmdcAvailable

Tests for mermaid CLI availability check.

Methods:

test_mmdc_available

```
def test_mmdc_available(mock_which)
```

Test when mmdc is available.

Parameter	Type	Default	Description
mock_which	-	-	-

`test_mmdc_not_available`

```
def test_mmdc_not_available(mock_which)
```

Test when mmdc is not available.

Parameter	Type	Default	Description
mock_which	-	-	-

`test_result_is_cached`

```
def test_result_is_cached(mock_which)
```

Test that the result is cached.

Parameter	Type	Default	Description
mock_which	-	-	-

class TestRenderMermaidToSvg

Tests for mermaid to SVG rendering.

Methods:

`test_returns_none_when_mmdc_unavailable`

```
def test_returns_none_when_mmdc_unavailable(mock_available)
```

Test that None is returned when mmdc is not available.

Parameter	Type	Default	Description
mock_available	-	-	-

test_renders_svg_successfully

```
def test_renders_svg_successfully(mock_available, mock_run, tmp_path)
```

Test successful SVG rendering.

Parameter	Type	Default	Description
mock_available	-	-	-
mock_run	-	-	-
tmp_path	-	-	-

test_returns_none_on_cli_error

```
def test_returns_none_on_cli_error(mock_available, mock_run)
```

Test that None is returned on CLI error.

Parameter	Type	Default	Description
mock_available	-	-	-
mock_run	-	-	-

test_handles_timeout

```
def test_handles_timeout(mock_available)
```

Test that timeout is handled gracefully.

Parameter	Type	Default	Description
mock_available	-	-	-

class TestRenderMermaidToPng

Tests for mermaid to PNG rendering.

Methods:**test_returns_none_when_mmdc_unavailable**

```
def test_returns_none_when_mmdc_unavailable(mock_available)
```

Test that None is returned when mmdc is not available.

Parameter	Type	Default	Description
mock_available	-	-	-

test_returns_none_on_cli_error

```
def test_returns_none_on_cli_error(mock_available, mock_run)
```

Test that None is returned on CLI error.

Parameter	Type	Default	Description
mock_available	-	-	-
mock_run	-	-	-

test_handles_timeout

```
def test_handles_timeout(mock_available)
```

Test that timeout is handled gracefully.

Parameter	Type	Default	Description
mock_available	-	-	-

class TestMermaidCliRendering

Tests for mermaid rendering with CLI available (uses PNG).

Methods:**test_renderer_mermaid_when_cli_available**

```
def test_renderer_mermaid_when_cli_available(mock_available, mock_render)
```

Test that mermaid diagrams are rendered as PNG when CLI is available.

Parameter	Type	Default	Description
mock_available	-	-	-
mock_render	-	-	-

test_falls_back_on_render_failure

```
def test_falls_back_on_render_failure(mock_available, mock_render)
```

Test fallback to placeholder when render fails.

Parameter	Type	Default	Description
mock_available	-	-	-
mock_render	-	-	-

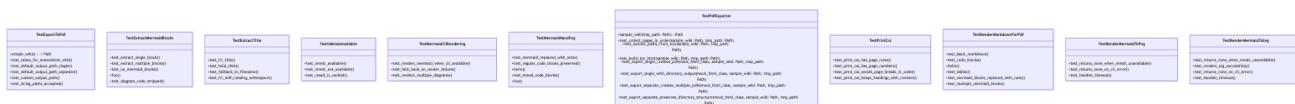
test_renderer_multiple_diagrams

```
def test_renderer_multiple_diagrams(mock_available, mock_render)
```

Test rendering multiple mermaid diagrams as PNG.

Parameter	Type	Default	Description
mock_available	-	-	-
mock_render	-	-	-

Class Diagram



Call Graph



Relevant Source Files

- tests/test_pdf_export.py:21–80

See Also

- [pdf](#) - dependency
 - [test_indexer](#) - shares 4 dependencies

test_retry.py

File Overview

This test module validates the retry functionality provided by the `with_retry` decorator and the `RETRYABLE_EXCEPTIONS` tuple from the `local_deepwiki.providers.base` module. The tests ensure that the retry mechanism properly handles various types of exceptions and respects configuration parameters.

Classes

TestWithRetry

The primary test class that validates the behavior of the `with_retry` decorator through comprehensive test scenarios.

Key Test Methods:

- `test_succeeds_on_first_attempt` - Verifies that successful functions execute normally without retries
- `test_retries_on_connection_error` - Tests retry behavior when `ConnectionError` is raised
- `test_retries_on_timeout_error` - Tests retry behavior when `TimeoutError` is raised
- `test_gives_up_after_max_attempts` - Validates that retries stop after reaching the maximum attempt limit
- `test_does_not_retry_non_retryable_errors` - Ensures non-retryable exceptions like `ValueError` are raised immediately
- `test_retries_on_rate_limit` - Tests retry behavior for rate limiting scenarios
- `test_retries_on_server_overload` - Tests retry behavior for server overload (503) errors
- `test_custom_max_attempts` - Validates that custom `max_attempts` parameter is respected

TestRetryableExceptions

A focused test class that validates the contents of the `RETRYABLE_EXCEPTIONS` tuple.

Test Methods:

- `test_includes_connection_error` - Verifies `ConnectionError` is in the retryable exceptions
- `test_includes_timeout_error` - Verifies `TimeoutError` is in the retryable exceptions
- `test_includes_os_error` - Verifies `OSError` is in the retryable exceptions

Usage Examples

Basic Retry Testing

```
@with_retry(max_attempts=3, base_delay=0.01)
async def flaky_func():
    # Function that may fail and needs retry
    if some_condition:
        raise ConnectionError("Connection refused")
    return "success"

# Test the retry behavior
result = await flaky_func()
assert result == "success"
```

Testing Non-Retryable Errors

```
@with_retry(max_attempts=3, base_delay=0.01)
async def value_error_func():
    raise ValueError("Invalid value")

# This will raise immediately without retries
with pytest.raises(ValueError):
    await value_error_func()
```

Custom Retry Configuration

```
@with_retry(max_attempts=5, base_delay=0.01)
async def func_with_custom_attempts():
    # Function with custom retry attempts
    pass
```

Related Components

- `local_deepwiki.providers.base.with_retry` - The retry decorator being tested
- `local_deepwiki.providers.base.RETRYABLE_EXCEPTIONS` - Tuple of exception types that trigger retries
- `pytest` - Testing framework used for assertions and exception handling

Test Configuration

All tests use minimal delay settings (`base_delay=0.01`) to ensure fast test execution while still validating the retry timing mechanisms. The tests cover both successful retry scenarios and failure cases where the maximum attempt limit is reached.

API Reference

class `TestWithRetry`

Tests for the `with_retry` decorator.

Methods:

`test_succeeds_on_first_attempt`

```
async def test_succeeds_on_first_attempt()
```

Test that successful calls work normally.

`successful_func`

```
async def successful_func()
```

`test_retries_on_connection_error`

```
async def test_retries_on_connection_error()
```

Test that connection errors trigger retry.

flaky_func

```
async def flaky_func()
```

test_retries_on_timeout_error

```
async def test_retries_on_timeout_error()
```

Test that timeout errors trigger retry.

timeout_func

```
async def timeout_func()
```

test_gives_up_after_max_attempts

```
async def test_gives_up_after_max_attempts()
```

Test that function gives up after max attempts.

always_fails

```
async def always_fails()
```

test_does_not_retry_non_retryable_errors

```
async def test_does_not_retry_non_retryable_errors()
```

Test that non-retryable errors are raised immediately.

value_error_func

```
async def value_error_func()
```

test_retries_on_rate_limit

```
async def test_retries_on_rate_limit()
```

Test that rate limit errors trigger retry.

rate_limited_func

```
async def rate_limited_func()
```

test_retries_on_server_overload

```
async def test_retries_on_server_overload()
```

Test that 503 errors trigger retry.

overloaded_func

```
async def overloaded_func()
```

test_preserves_function_metadata

```
async def test_preserves_function_metadata()
```

Test that decorator preserves function name and docstring.

documented_func

```
async def documented_func()
```

This is a docstring.

test_custom_max_attempts

```
async def test_custom_max_attempts()
```

Test that max_attempts parameter is respected.

func_with_custom_attempts

```
async def func_with_custom_attempts()
```

class TestRetryableExceptions

Tests for the RETRYABLE_EXCEPTIONS tuple.

Methods:

test_includes_connection_error

```
def test_includes_connection_error()
```

Test that ConnectionError is retryable.

test_includes_timeout_error

```
def test_includes_timeout_error()
```

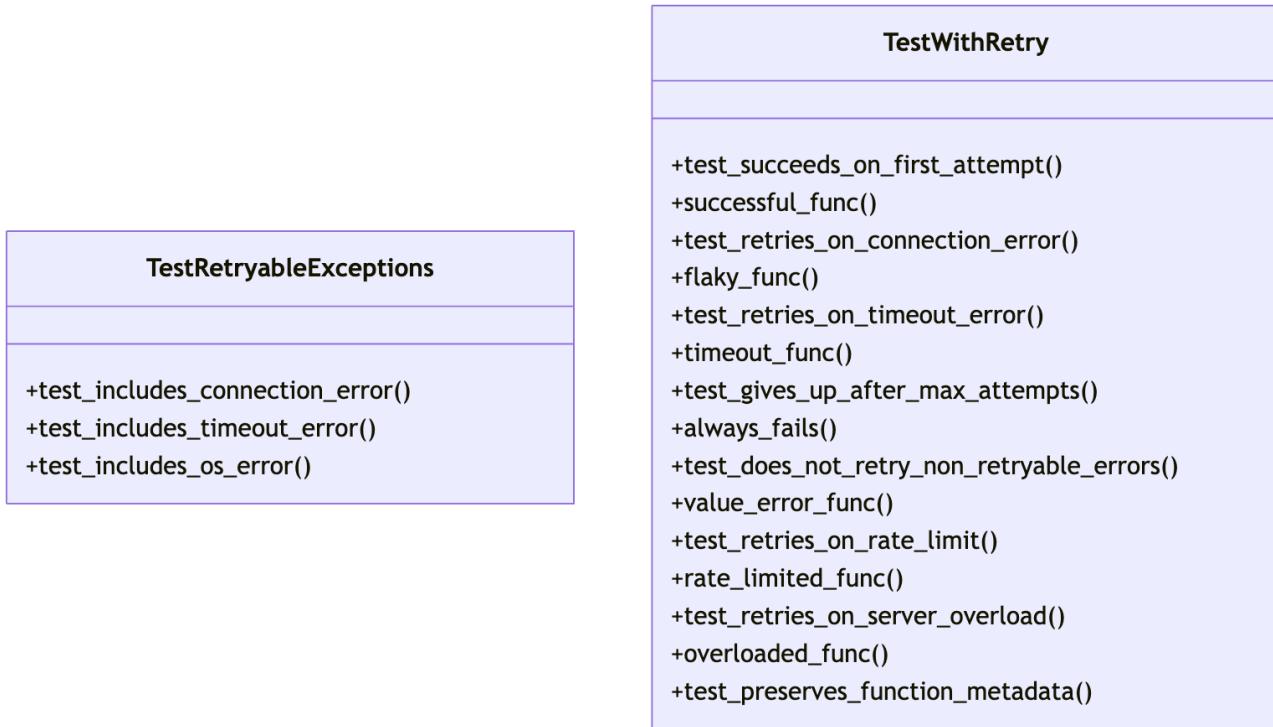
Test that TimeoutError is retryable.

test_includes_os_error

```
def test_includes_os_error()
```

Test that OSError is retryable.

Class Diagram



Call Graph



Relevant Source Files

- tests/test_retry.py:8-144

See Also

- [test_vectorstore](#) - shares 2 dependencies

test_vectorstore.py

File Overview

This file contains comprehensive test suites for the [VectorStore](#) component, covering search functionality, index operations, statistics, chunk management, and edge cases. The tests ensure the vector store correctly handles data operations, SQL injection attempts, and various edge conditions.

Classes

MockEmbeddingProvider

A mock implementation of the EmbeddingProvider interface used for testing purposes. This class provides fake embeddings to isolate vector store functionality from actual embedding generation.

TestVectorStoreIndexes

Tests for vector store scalar indexes functionality. This test class verifies that the vector store correctly manages and utilizes indexes for efficient data retrieval.

Key Methods: - `vector_store` : Fixture that creates a [VectorStore](#) instance with a temporary database path and MockEmbeddingProvider - `populated_store` : Fixture that creates a vector store with test data including chunks from "src/main.py"

TestVectorStoreSearch

Tests for vector store search functionality. This class validates that search operations work correctly across different scenarios.

Key Methods: - `vector_store` : Fixture that creates a [VectorStore](#) instance for testing - `test_search_empty_store` : Verifies that searching an empty store returns empty results

TestVectorStoreStats

Tests for vector store statistics functionality. This class ensures that the statistics reporting features work correctly.

Key Methods: - `vector_store` : Fixture that creates a [VectorStore](#) instance for testing -
`test_stats_empty_store` : Tests that statistics for an empty store return correct values (total_chunks: 0, languages: {})

TestVectorStoreAddChunks

Tests for adding chunks to existing tables. This class verifies that the vector store correctly handles adding new chunks to the database.

Key Methods: - `vector_store` : Fixture that creates a [VectorStore](#) instance for testing -
`test_add_to_emptyCreatesTable` : Tests that adding chunks to an empty store creates the necessary table structure

TestVectorStoreEdgeCases

Comprehensive tests for edge cases and error conditions. This class covers security concerns, data integrity, and boundary conditions.

Key Methods: - `vector_store` : Fixture that creates a [VectorStore](#) instance for testing -
`test_get_chunk_by_id_empty_db` : Tests that retrieving a chunk by ID from an empty database returns None - `test_get_chunks_by_file_empty_db` : Tests that retrieving chunks by file from an empty database returns an empty list - `test_delete_chunks_by_file_empty_db` : Tests that deleting chunks by file from an empty database returns 0 - `test_create_or_update_empty_list` : Tests that creating or updating with an empty list returns 0 and maintains empty statistics -
`test_file_path_with_quotes` : Tests that file paths containing quotes are handled safely without causing injection issues - `test_delete_file_path_with_quotes` : Tests that deleting files with quotes in their paths works correctly - `test_file_path_injection_attempt` : Tests that SQL-like injection attempts in file paths are neutralized and don't cause errors - `test_search_limit_zero_raises` : Tests that searching with limit=0 raises a ValueError as required by LanceDB

Usage Examples

Creating a Test Vector Store

```
@pytest.fixture
def vector_store(self, tmp_path):
    from local_deepwiki.core.vectorstore import VectorStore

    db_path = tmp_path / "test.lance"
    provider = MockEmbeddingProvider()
    return VectorStore(db_path, provider)
```

Testing Empty Store Statistics

```
def test_stats_empty_store(self, vector_store):
    stats = vector_store.get_stats()
    assert stats["total_chunks"] == 0
    assert stats["languages"] == {}
```

Testing Search with Invalid Limit

```
async def test_search_limit_zero_raises(self, vector_store):
    chunk = make_chunk("test")
    await vector_store.create_or_update_table([chunk])

    with pytest.raises(ValueError, match="Limit is required"):
        await vector_store.search("test", limit=0)
```

Related Components

This test file works with several components from the `local_deepwiki` package:

- **VectorStore**: The `main` class being tested, imported from `local_deepwiki.core.vectorstore`
- **EmbeddingProvider**: Base class from `local_deepwiki.providers.base` that `MockEmbeddingProvider` implements
- **CodeChunk**: Model from `local_deepwiki.models` used for creating test data
- **ChunkType and Language**: Enums from `local_deepwiki.models` used in chunk creation

The tests use pytest fixtures and async/await patterns, indicating the `VectorStore` operations are asynchronous and the codebase uses modern Python async functionality.

API Reference

`class MockEmbeddingProvider`

Inherits from: `EmbeddingProvider`

Mock embedding provider for testing.

Methods:

`__init__`

```
def __init__(dimension: int = 384)
```

Parameter	Type	Default	Description
dimension	int	384	-

`name`

```
def name() -> str
```

Return provider name.

`get_dimension`

```
def get_dimension() -> int
```

Return embedding dimension.

`embed`

```
async def embed(texts: list[str]) -> list[list[float]]
```

Generate mock embeddings.

Parameter	Type	Default	Description
texts	list[str]	-	-

class TestVectorStoreIndexes

Tests for vector store scalar indexes.

Methods:

vector_store

```
def vector_store(tmp_path)
```

Create a vector store for testing.

Parameter	Type	Default	Description
tmp_path	-	-	-

populated_store

```
async def populated_store(vector_store)
```

Create a vector store with test data.

Parameter	Type	Default	Description
vector_store	-	-	-

test_create_tableCreates_indexes

```
async def test_create_tableCreates_indexes(populated_store)
```

Test that creating a table creates scalar indexes.

Parameter	Type	Default	Description
populated_store	-	-	-

test_get_chunk_by_id_uses_index

```
async def test_get_chunk_by_id_uses_index(populated_store)
```

Test that get_chunk_by_id can [find](#) chunks efficiently.

Parameter	Type	Default	Description
populated_store	-	-	-

test_get_chunks_by_file_uses_index

```
async def test_get_chunks_by_file_uses_index(populated_store)
```

Test that get_chunks_by_file can [find](#) chunks efficiently.

Parameter	Type	Default	Description
populated_store	-	-	-

test_delete_chunks_by_file_uses_index

```
async def test_delete_chunks_by_file_uses_index(populated_store)
```

Test that delete_chunks_by_file works efficiently.

Parameter	Type	Default	Description
populated_store	-	-	-

test_ensure_indexes_on_existing_table

```
async def test_ensure_indexes_on_existing_table(vector_store, tmp_path)
```

Test that opening an existing table ensures indexes exist.

Parameter	Type	Default	Description
vector_store	-	-	-
tmp_path	-	-	-

class TestVectorStoreSearch

Tests for vector store search functionality.

Methods:

vector_store

```
def vector_store(tmp_path)
```

Create a vector store for testing.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_search_empty_store

```
async def test_search_empty_store(vector_store)
```

Test searching an empty store returns empty results.

Parameter	Type	Default	Description
vector_store	-	-	-

test_search_with_results

```
async def test_search_with_results(vector_store)
```

Test searching returns results.

Parameter	Type	Default	Description
vector_store	-	-	-

`test_search_with_language_filter`

```
async def test_search_with_language_filter(vector_store)
```

Test searching with language filter.

Parameter	Type	Default	Description
vector_store	-	-	-

`test_search_invalid_language_raises`

```
async def test_search_invalid_language_raises(vector_store)
```

Test searching with invalid language raises ValueError.

Parameter	Type	Default	Description
vector_store	-	-	-

`test_search_with_chunk_type_filter`

```
async def test_search_with_chunk_type_filter(vector_store)
```

Test searching with chunk type filter.

Parameter	Type	Default	Description
vector_store	-	-	-

test_search_invalid_chunk_type_raises

```
async def test_search_invalid_chunk_type_raises(vector_store)
```

Test searching with invalid chunk type raises ValueError.

Parameter	Type	Default	Description
vector_store	-	-	-

class TestVectorStoreStats

Tests for vector store statistics.

Methods:**vector_store**

```
def vector_store(tmp_path)
```

Create a vector store for testing.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_stats_empty_store

```
def test_stats_empty_store(vector_store)
```

Test stats for empty store.

Parameter	Type	Default	Description
vector_store	-	-	-

test_stats_with_data

```
async def test_stats_with_data(vector_store)
```

Test stats with data.

Parameter	Type	Default	Description
vector_store	-	-	-

class TestVectorStoreAddChunks

Tests for adding chunks to existing table.

Methods:**vector_store**

```
def vector_store(tmp_path)
```

Create a vector store for testing.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_add_to_emptyCreatesTable

```
async def test_add_to_emptyCreatesTable(vector_store)
```

Test adding to empty store creates table.

Parameter	Type	Default	Description
vector_store	-	-	-

test_add_to_existing_table

```
async def test_add_to_existing_table(vector_store)
```

Test adding chunks to existing table.

Parameter	Type	Default	Description
vector_store	-	-	-

test_add_empty_list

```
async def test_add_empty_list(vector_store)
```

Test adding empty list returns 0.

Parameter	Type	Default	Description
vector_store	-	-	-

class TestVectorStoreEdgeCases

Tests for vector store edge cases and error handling.

Methods:**vector_store**

```
def vector_store(tmp_path)
```

Create a vector store for testing.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_get_chunk_by_id_empty_db

```
async def test_get_chunk_by_id_empty_db(vector_store)
```

Test get_chunk_by_id on empty database returns None.

Parameter	Type	Default	Description
vector_store	-	-	-

test_get_chunks_by_file_empty_db

```
async def test_get_chunks_by_file_empty_db(vector_store)
```

Test get_chunks_by_file on empty database returns empty list.

Parameter	Type	Default	Description
vector_store	-	-	-

test_delete_chunks_by_file_empty_db

```
async def test_delete_chunks_by_file_empty_db(vector_store)
```

Test delete_chunks_by_file on empty database returns 0.

Parameter	Type	Default	Description
vector_store	-	-	-

test_create_or_update_empty_list

```
async def test_create_or_update_empty_list(vector_store)
```

Test create_or_update_table with empty list returns 0.

Parameter	Type	Default	Description
vector_store	-	-	-

test_chunk_id_with_quotes

```
async def test_chunk_id_with_quotes(vector_store)
```

Test chunk ID with single quotes is handled safely.

Parameter	Type	Default	Description
vector_store	-	-	-

test_file_path_with_quotes

```
async def test_file_path_with_quotes(vector_store)
```

Test file path with quotes is handled safely.

Parameter	Type	Default	Description
vector_store	-	-	-

test_delete_file_path_with_quotes

```
async def test_delete_file_path_with_quotes(vector_store)
```

Test deleting file path with quotes is handled safely.

Parameter	Type	Default	Description
vector_store	-	-	-

test_chunk_id_injection_attempt

```
async def test_chunk_id_injection_attempt(vector_store)
```

Test that SQL-like injection in chunk_id is neutralized.

Parameter	Type	Default	Description
vector_store	-	-	-

test_file_path_injection_attempt

```
async def test_file_path_injection_attempt(vector_store)
```

Test that SQL-like injection in file_path is neutralized.

Parameter	Type	Default	Description
vector_store	-	-	-

test_unicode_content

```
async def test_unicode_content(vector_store)
```

Test handling of Unicode content in chunks.

Parameter	Type	Default	Description
vector_store	-	-	-

test_reopen_database

```
async def test_reopen_database(tmp_path)
```

Test reopening database preserves data.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_replace_existing_table

```
async def test_replace_existing_table(vector_store)
```

Test create_or_update_table replaces existing data.

Parameter	Type	Default	Description
vector_store	-	-	-

test_db_path_created_if_not_exists

```
async def test_db_path_created_if_not_exists(tmp_path)
```

Test that database directory is created if it doesn't exist.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_single_chunk_operations

```
async def test_single_chunk_operations(vector_store)
```

Test operations with single chunk.

Parameter	Type	Default	Description
vector_store	-	-	-

test_empty_content_chunk

```
async def test_empty_content_chunk(vector_store)
```

Test chunk with empty content.

Parameter	Type	Default	Description
vector_store	-	-	-

test_large_content_chunk

```
async def test_large_content_chunk(vector_store)
```

Test chunk with large content.

Parameter	Type	Default	Description
vector_store	-	-	-

test_many_chunks_same_file

```
async def test_many_chunks_same_file(vector_store)
```

Test many chunks from same file.

Parameter	Type	Default	Description
vector_store	-	-	-

test_search_limit_zero_raises

```
async def test_search_limit_zero_raises(vector_store)
```

Test search with limit=0 raises ValueError.

Parameter	Type	Default	Description
vector_store	-	-	-

test_search_very_long_query

```
async def test_search_very_long_query(vector_store)
```

Test search with very long query string.

Parameter	Type	Default	Description
vector_store	-	-	-

Functions

make_chunk

```
def make_chunk(id: str, file_path: str = "test.py", content: str = "test code", language: Language =
```

Create a test code chunk.

Parameter	Type	Default	Description
id	str	-	-
file_path	str	"test.py"	-
content	str	"test code"	-
language	Language	Language.PYTHON	-
chunk_type	ChunkType	ChunkType.FUNCTION	-

Returns: CodeChunk

Class Diagram



Call Graph



Relevant Source Files

- `tests/test_vectorstore.py:9–28`

See Also

- [vectorstore](#) - dependency
- [test_api_docs](#) - shares 2 dependencies

File Overview

This file defines the core chunking functionality for processing code files. It provides the `CodeChunker` class responsible for breaking down code into meaningful chunks based on language-specific parsing rules. The chunker identifies different types of code elements such as functions, classes, and methods, and organizes them into structured `CodeChunk` objects.

Classes

CodeChunker

The `CodeChunker` class is responsible for parsing code files and splitting them into logical chunks based on the structure of the code. It uses tree-sitter for parsing and supports multiple programming languages.

Key Methods

- `chunk_file(self, file_path: Path) -> Iterator[CodeChunk]` : Takes a file path and yields `CodeChunk` objects representing different parts of the code file.
- `chunk_node(self, node: Node, file_path: Path, lang: Language) -> Iterator[CodeChunk]` : Processes a tree-sitter node and generates chunks for it, recursively handling child nodes.

Functions

get_parent_classes

```
def get_parent_classes(node: Node, lang: Language) -> list[str]
```

Retrieves the names of parent classes for a given node in the code structure.

Parameters

- `node` : A tree-sitter Node object representing a class or similar construct
- `lang` : Language enum indicating the programming language of the code

Returns

- A list of strings representing the names of parent classes

Usage Examples

```
from pathlib import Path
from local_deepwiki.core.chunker import CodeChunker

# Initialize the chunker
chunker = CodeChunker()

# Process a file
file_path = Path("example.py")
for chunk in chunker.chunk_file(file_path):
    print(f"Chunk type: {chunk.type}, Content: {chunk.content}")
```

Related Components

This file works with the following components based on imports:

- **ChunkingConfig**: Configuration settings for chunking behavior
- **CodeParser**: Parser for code files using tree-sitter
- **get_node_text, get_node_name, get_docstring, find_nodes_by_type**: Helper functions for parsing code nodes
- **CodeChunk, ChunkType, Language**: Data models for representing code chunks and their properties
- **get_config**: Function to retrieve configuration settings

API Reference

class CodeChunker

Extract semantic code chunks from source files using AST analysis.

Methods:

__init__

```
def __init__(config: ChunkingConfig | None = None)
```

Initialize the chunker.

Parameter	Type	Default	Description
config	ChunkingConfig None	None	Optional chunking configuration.

chunk_file

```
def chunk_file(file_path: Path, repo_root: Path) -> Iterator[CodeChunk]
```

Extract code chunks from a source file.

Parameter	Type	Default	Description
file_path	Path	-	Path to the source file.
repo_root	Path	-	Root directory of the repository.

Functions

get_parent_classes

```
def get_parent_classes(class_node: Node, source: bytes, language: Language) -> list[str]
```

Extract parent class names from a class definition.

Parameter	Type	Default	Description
class_node	Node	-	The class AST node.
source	bytes	-	Source bytes.
language	Language	-	Programming language.

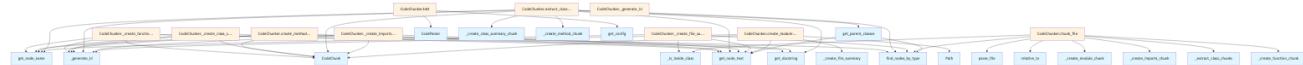
Returns: list[str]

Class Diagram

CodeChunker

- init()**
- +**chunk_file()**
- _create_module_chunk()**
- _create_file_summary()**
- _create_imports_chunk()**
- _extract_class_chunks()**
- _create_class_summary_chunk()**
- _create_method_chunk()**
- _create_function_chunk()**
- _is_inside_class()**
- _generate_id()**

Call Graph



Relevant Source Files

- [src/local_deepwiki/core/chunker.py:174–562](#)

See Also

- [api_docs](#) - uses this
- [config](#) - dependency
- [parser](#) - dependency
- [wiki](#) - shares 5 dependencies

Indexer Module Documentation

File Overview

The indexer.py file is responsible for indexing code repositories by parsing source files, chunking code into manageable segments, and storing embeddings in a vector database. It serves as the core component that transforms raw code into searchable semantic representations. This module works closely with the chunker to break down code into chunks, the parser to understand code structure, and the vectorstore to persist embeddings for similarity search.

The indexer integrates with the configuration system to determine indexing behavior and works with embedding providers to generate semantic representations of code chunks. It coordinates progress tracking through the rich library for user feedback during long-running indexing operations.

Classes

RepositoryIndexer

The RepositoryIndexer class orchestrates the complete indexing process for a code repository. It handles parsing source files, chunking code into semantic units, generating embeddings, and storing these embeddings in a vector database.

Key methods:

- `index_repository` : Main method that performs the complete indexing workflow
- `process_file` : Handles individual file processing including parsing and chunking
- `update_index_status` : Updates the indexing status for tracking progress

Usage Example:

```
from local_deepwiki.core.indexer import RepositoryIndexer
from local_deepwiki.config import get_config

config = get_config()
indexer = RepositoryIndexer(config)
indexer.index_repository()
```

Functions

index_repository

The index_repository function is the [main](#) entry point for the indexing process. It iterates through all files matching the configured patterns, processes each file through the parsing and chunking pipeline, and stores the resulting embeddings in the vector database.

Parameters: - config ([Config](#)): Configuration object containing indexing settings
- progress (Progress): Progress tracking object for user feedback
- task_id (TaskID): Current task identifier for progress updates

Return Value: - None

Usage Example:

```
from local_deepwiki.core.indexer import index_repository
from local_deepwiki.config import get_config

config = get_config()
index_repository(config)
```

process_file

The process_file function handles the processing of individual files within the repository. It parses the file content, chunks it into semantic units, and generates embeddings for each chunk.

Parameters: - file_path (Path): Path to the file being processed
- config ([Config](#)): Configuration object for indexing behavior
- progress (Progress): Progress tracking object
- task_id (TaskID): Current task identifier

Return Value: - List[[CodeChunk](#)]: List of code chunks generated from the file

update_index_status

The update_index_status function updates the indexing status for tracking progress and monitoring the indexing workflow.

Parameters: - file_path (Path): Path to the file being processed - status ([IndexStatus](#)): Current indexing status - progress (Progress): Progress tracking object - task_id (TaskID): Current task identifier

Return Value: - None

Usage Examples

Basic Indexing Workflow

```
from local_deepwiki.core.indexer import RepositoryIndexer
from local_deepwiki.config import get_config

# Get configuration
config = get_config()

# Initialize indexer
indexer = RepositoryIndexer(config)

# Start indexing
indexer.index_repository()
```

Custom Indexing with Progress Tracking

```
from local_deepwiki.core.indexer import RepositoryIndexer
from local_deepwiki.config import get_config
from rich.progress import Progress

config = get_config()
indexer = RepositoryIndexer(config)

with Progress() as progress:
    task = progress.add_task("Indexing repository...", total=100)
    indexer.index_repository(progress, task)
```

Related Components

This class works with [VectorStore](#) to store embeddings and retrieve similar code chunks. It integrates with [CodeChunker](#) to break down source code into semantic units and with [CodeParser](#) to understand code structure. The indexer relies on `get_embedding_provider` to generate semantic representations of code chunks. It also uses the [Config](#) system to determine indexing patterns and behavior, and the [FileInfo](#) model to track file information during indexing.

API Reference

class `RepositoryIndexer`

Orchestrates repository indexing with incremental update support.

Methods:

`__init__`

```
def __init__(repo_path: Path, config: Config | None = None, embedding_provider_name: str | None = No
```

Initialize the indexer.

Parameter	Type	Default	Description
<code>repo_path</code>	<code>Path</code>	-	Path to the repository root.
<code>config</code>	<code>Config None</code>	<code>None</code>	Optional configuration.
<code>embedding_provider_name</code>	<code>str None</code>	<code>None</code>	Override embedding provider ("local" or "openai").

`index`

```
async def index(full_rebuild: bool = False, progress_callback: Callable[[str, int, int], None] | Non
```

Index the repository.

Parameter	Type	Default	Description
full_rebuild	bool	False	If True, rebuild entire index. Otherwise, incremental update.
progress_callback	Callable[[str, int, int], None] None	None	Optional callback for progress updates (message, current, total).

get_status

```
def get_status() -> IndexStatus | None
```

Get the current indexing status.

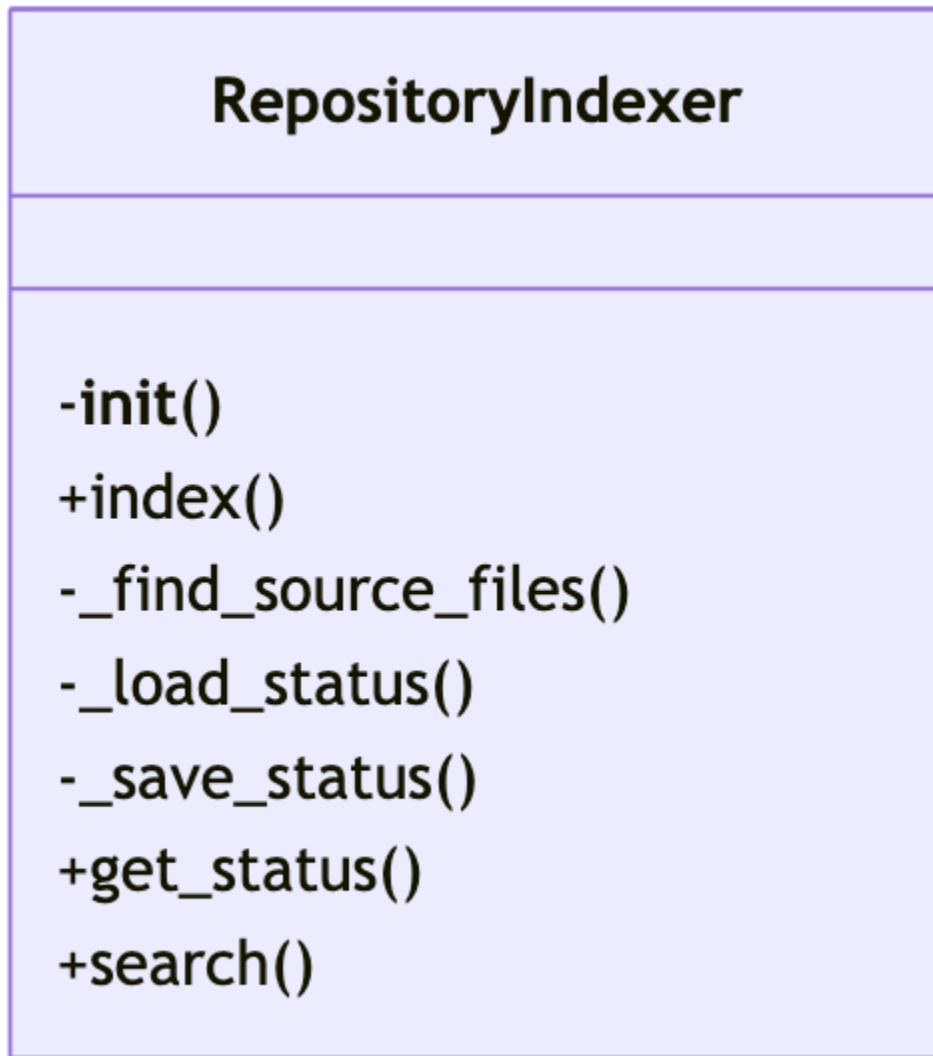
search

```
async def search(query: str, limit: int = 10, language: str | None = None) -> list[dict]
```

Search the indexed repository.

Parameter	Type	Default	Description
query	str	-	Search query.
limit	int	10	Maximum results.
language	str None	None	Optional language filter.

Class Diagram



Call Graph



See Also

- [server](#) - uses this
- [watcher](#) - uses this
- [chunker](#) - dependency
- [models](#) - dependency
- [config](#) - dependency

File Overview

This file, `src/local_deepwiki/export/html.py`, provides functionality for exporting a DeepWiki documentation set to static HTML files. It includes a class `HtmlExporter` for handling the export process and utility functions for rendering markdown content and managing the CLI interface.

Classes

HtmlExporter

The `HtmlExporter` class is responsible for exporting all markdown pages from a DeepWiki directory to static HTML files.

Methods

```
__init__(self, wiki_path: Path, output_path: Path)
```

Initialize the exporter.

Parameters: - `wiki_path` : Path to the `.deepwiki` directory - `output_path` : Output directory for HTML files

```
export(self) -> int
```

Export all wiki pages to HTML.

Returns: - Number of pages exported

```
_export_page(self, md_file: Path, rel_path: Path) -> None
```

Export a single markdown page to HTML.

Parameters: - `md_file` : Path to the markdown file - `rel_path` : Relative path from wiki root

```
_render_toc(self) -> str
```

Render the table of contents as HTML.

```
_render_toc_entry(self, entry: dict) -> str
```

Render a single table of contents entry as HTML.

```
_build_breadcrumb(self, rel_path: Path) -> str
```

Build a breadcrumb navigation for a page.

Functions

```
render_markdown(content: str) -> str
```

Convert markdown content to HTML.

Parameters: - `content` : Markdown text to convert

Returns: - HTML string

```
extract_title(md_file: Path) -> str
```

Extract the title from a markdown file.

Parameters: - `md_file` : Path to the markdown file

Returns: - Title string

```
export_to_html(wiki_path: str | Path, output_path: str | Path | None = None) -> str
```

Export wiki to static HTML files.

Parameters: - `wiki_path` : Path to the .deepwiki directory - `output_path` : Output directory (default: `{wiki_path}_html`)

Returns: - Path to the output directory

```
main()
```

CLI entry point for HTML export.

Usage Examples

Using `export_to_html` function

```
from src.local_deepwiki.export.html import export_to_html

# Export to default output directory
output_dir = export_to_html(".deepwiki")

# Export to custom output directory
output_dir = export_to_html(".deepwiki", "output/html")
```

Using `HtmlExporter` class directly

```
from pathlib import Path
from src.local_deepwiki.export.html import HtmlExporter

exporter = HtmlExporter(
    wiki_path=Path(".deepwiki"),
    output_path=Path("output/html")
)
count = exporter.export()
```

CLI usage

```
# Export to default output directory
python -m src.local_deepwiki.export.html

# Export to custom output directory
python -m src.local_deepwiki.export.html --output custom_output

# Export from specific wiki path
python -m src.local_deepwiki.export.html /path/to/wiki
```

Related Components

This file imports and uses: - `argparse` for command-line interface handling - `json` for parsing TOC and search data - `shutil` for copying files - `pathlib.Path` for path manipulation - `markdown` for markdown rendering

The `HtmlExporter` class works with: - `toc.json` file for table of contents data - `search.json` file for search functionality - Markdown files in the wiki directory - HTML template files (not shown in this file)

API Reference

`class HtmlExporter`

Export wiki markdown to static HTML files.

Methods:

`__init__`

```
def __init__(wiki_path: Path, output_path: Path)
```

Initialize the exporter.

Parameter	Type	Default	Description
wiki_path	Path	-	Path to the .deepwiki directory
output_path	Path	-	Output directory for HTML files

export

```
def export() -> int
```

Export all wiki pages to HTML.

Functions

render_markdown

```
def render_markdown(content: str) -> str
```

Render markdown to HTML.

Parameter	Type	Default	Description
content	str	-	-

Returns: str

extract_title

```
def extract_title(md_file: Path) -> str
```

Extract title from markdown file.

Parameter	Type	Default	Description
md_file	Path	-	-

Returns: str

`export_to_html`

```
def export_to_html(wiki_path: str | Path, output_path: str | Path | None = None) -> str
```

Export wiki to static HTML files.

Parameter	Type	Default	Description
wiki_path	str Path	-	Path to the .deepwiki directory
output_path	str Path None	None	Output directory (default: {wiki_path}_html)

Returns: str

`main`

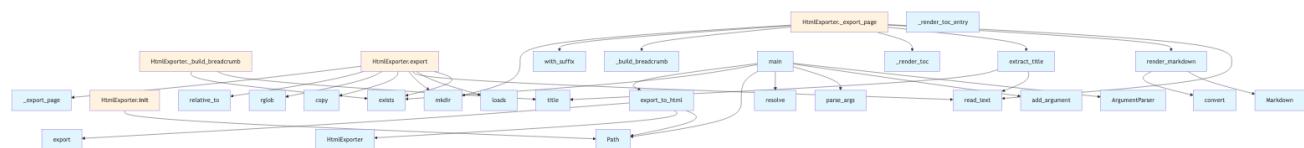
```
def main()
```

CLI entry point for HTML export.

Class Diagram



Call Graph



Relevant Source Files

- `src/local_deepwiki/export/html.py:654-841`

See Also

- [server](#) - uses this
- [test_html_export](#) - uses this

Search Index Generator

File Overview

This file provides functionality for generating search indexes from wiki pages. It extracts relevant content from wiki pages including headings, code terms, and snippets to create a searchable index that can be used for fast lookup and filtering of wiki content.

Dependencies

- `json` - For serializing the search index to JSON format
- `re` - For regular expression operations when extracting content
- `pathlib.Path` - For handling file system paths
- `local_deepwiki.models.WikiPage` - Model representing wiki pages

Functions

`extract_headings(content: str) -> list[str]`

Extracts all headings from the given content.

Parameters: - `content` (str): The wiki page content to extract headings from

Returns: - `list[str]` : List of extracted headings

`extract_code_terms(content: str) -> list[str]`

Extracts code terms (likely code snippets or identifiers) from the given content.

Parameters: - `content` (str): The wiki page content to extract code terms from

Returns: - `list[str]` : List of extracted code terms

`extract_snippet(content: str) -> str`

Extracts a snippet from the given content, likely the first paragraph or a summary.

Parameters: - `content` (str): The wiki page content to extract snippet from

Returns: - `str` : Extracted snippet text

`generate_search_entry(page: WikiPage) -> dict`

Generate a search index entry for a wiki page.

Parameters: - `page` (WikiPage): The wiki page to generate entry for

Returns: - `dict` : Dictionary with searchable fields including path, title, headings, terms, and snippet

`generate_search_index(pages: list[WikiPage]) -> list[dict]`

Generate a search index from wiki pages.

Parameters: - `pages` (list[WikiPage]): List of wiki pages to index

Returns: - `list[dict]` : List of search entries for all pages

`write_search_index(wiki_path: Path, pages: list[WikiPage]) -> Path`

Generate and write search index to disk.

Parameters: - `wiki_path` (Path): Path to wiki directory where index should be written - `pages` (list[WikiPage]): List of wiki pages to index

Returns: - `Path` : Path to the generated search.json file

Usage Examples

```
from pathlib import Path
from local_deepwiki.models import WikiPage
from local_deepwiki.generators.search import write_search_index

# Example usage
wiki_path = Path("/path/to/wiki")
pages = [
    WikiPage(
        path="page1.md",
        title="Page One",
        content="# Introduction\nThis is the content of page one."
    ),
    WikiPage(
        path="page2.md",
        title="Page Two",
        content="# Guide\nThis is the content of page two."
    )
]

# Generate and write search index
index_path = write_search_index(wiki_path, pages)
print(f"Search index written to: {index_path}")
```

Output Format

The generated search index is a JSON file containing an array of entries, each with the following structure:

```
[
  {
    "path": "page1.md",
    "title": "Page One",
    "headings": ["Introduction"],
    "terms": ["code", "example"],
    "snippet": "This is the content of page one."
  }
]
```

See Also

- [test_search](#) - uses this
- [wiki](#) - uses this
- [models](#) - dependency
- [crosslinks](#) - shares 3 dependencies
- [see_also](#) - shares 3 dependencies

File Overview

This file, `see_also.py`, is part of the `local_deepwiki` package and provides functionality for generating "See Also" sections in wiki pages. It helps establish relationships between source files and their corresponding wiki documentation by mapping file paths and analyzing dependencies to suggest related content.

The module includes utilities for building mappings from source files to wiki pages, generating see-also content, and adding these sections to wiki pages.

Classes

FileRelationships

A dataclass used to store information about file relationships.

Fields

- `source_file` (`str`): The path of the source file.
- `wiki_page` (`str`): The corresponding wiki page path.
- `related_files` (`list[str]`): A list of related file paths.

RelationshipAnalyzer

A class responsible for analyzing relationships between source files and wiki pages.

Methods

- `analyze() → dict[str, list[str]]` : Analyzes the relationships and returns a dictionary mapping source files to lists of related files.

Functions

build_file_to_wiki_map

```
def build_file_to_wiki_map(pages: list[WikiPage]) -> dict[str, str]:
```

Builds a mapping from source file paths to wiki page paths.

Parameters

- `pages` (`list[WikiPage]`): A list of wiki pages to process.

Returns

- `dict[str, str]` : A dictionary mapping source file paths to wiki page paths.

generate_see_also_section

```
def generate_see_also_section(related_files: list[str]) -> str:
```

Generates a formatted "See Also" section for a wiki page based on a list of related files.

Parameters

- `related_files` (`list[str]`): A list of file paths related to the current page.

Returns

- `str` : A formatted markdown string representing the "See Also" section.

_relative_path

```
def _relative_path(file_path: str, base_path: str) -> str:
```

Computes the relative path of a file with respect to a base path.

Parameters

- `file_path` (str): The absolute or full file path.
- `base_path` (str): The base path to compute the relative path from.

Returns

- `str` : The relative path of `file_path` with respect to `base_path`.

add_see_also_sections

```
def add_see_also_sections(wiki_pages: list[WikiPage], file_to_wiki_map: dict[str, str]) -> None:
```

Adds "See Also" sections to the content of wiki pages based on the file-to-wiki mapping.

Parameters

- `wiki_pages` (list[WikiPage]): A list of wiki pages to update.
- `file_to_wiki_map` (dict[str, str]): A mapping from source file paths to wiki page paths.

Returns

- `None` : This function modifies the wiki pages in place.

Usage Examples

Building a File-to-Wiki Map

```
from local_deepwiki.generators.see_also import build_file_to_wiki_map
from local_deepwiki.models import WikiPage

pages = [WikiPage(path="files/src/local_deepwiki/core/chunker.md", content="...")]
file_to_wiki_map = build_file_to_wiki_map(pages)
```

Generating a See Also Section

```
from local_deepwiki.generators.see_also import generate_see_also_section

related_files = ["src/local_deepwiki/core/chunker.py", "src/local_deepwiki/core/parser.py"]
section = generate_see_also_section(related_files)
```

Adding See Also Sections to Wiki Pages

```
from local_deepwiki.generators.see_also import add_see_also_sections
from local_deepwiki.models import WikiPage

wiki_pages = [WikiPage(path="files/src/local_deepwiki/core/chunker.md", content="...")]
file_to_wiki_map = {"src/local_deepwiki/core/chunker.py": "files/src/local_deepwiki/core/chunker.md"}

add_see_also_sections(wiki_pages, file_to_wiki_map)
```

Related Components

This module interacts with the following components:

- `WikiPage` from `local_deepwiki.models` : Represents a wiki page with a path and content.

- `ChunkType` and `CodeChunk` from `local_deepwiki.models` : Used for handling code chunks and their types, although not directly used in this file.
- `Path` from `pathlib` : Used for handling file paths.
- `re` : Used for regular expressions, although not directly used in this file.

API Reference

class `FileRelationships`

Relationships for a single file.

class `RelationshipAnalyzer`

Analyzes import relationships between source files. This class builds a graph of file dependencies from import chunks, enabling discovery of related files through various relationship types.

Methods:

`__init__`

```
def __init__() -> None
```

Initialize an empty relationship analyzer.

`analyze_chunks`

```
def analyze_chunks(chunks: list[CodeChunk]) -> None
```

Analyze import chunks to build relationship graph.

Parameter	Type	Default	Description
<code>chunks</code>	<code>list[CodeChunk]</code>	-	List of code chunks (should include IMPORT chunks).

get_relationships

```
def get_relationships(file_path: str) -> FileRelationships
```

Get all relationships for a file.

Parameter	Type	Default	Description
file_path	str	-	Path to the source file.

get_all_known_files

```
def get_all_known_files() -> set[str]
```

Get all known file paths.

Functions

build_file_to_wiki_map

```
def build_file_to_wiki_map(pages: list[WikiPage]) -> dict[str, str]
```

Build a mapping from source file paths to wiki page paths.

Parameter	Type	Default	Description
pages	list[WikiPage]	-	List of wiki pages.

Returns: dict[str, str]

generate_see_also_section

```
def generate_see_also_section(relationships: FileRelationships, file_to_wiki: dict[str, str], current
```

Generate a See Also section for a wiki page.

Parameter	Type	Default	Description
relationships	FileRelationships	-	The file relationships.
file_to_wiki	dict[str, str]	-	Mapping of source files to wiki paths.
current_wiki_path	str	-	Path of the current wiki page.
max_items	int	5	Maximum number of items to include.

Returns: str | None

add_see_also_sections

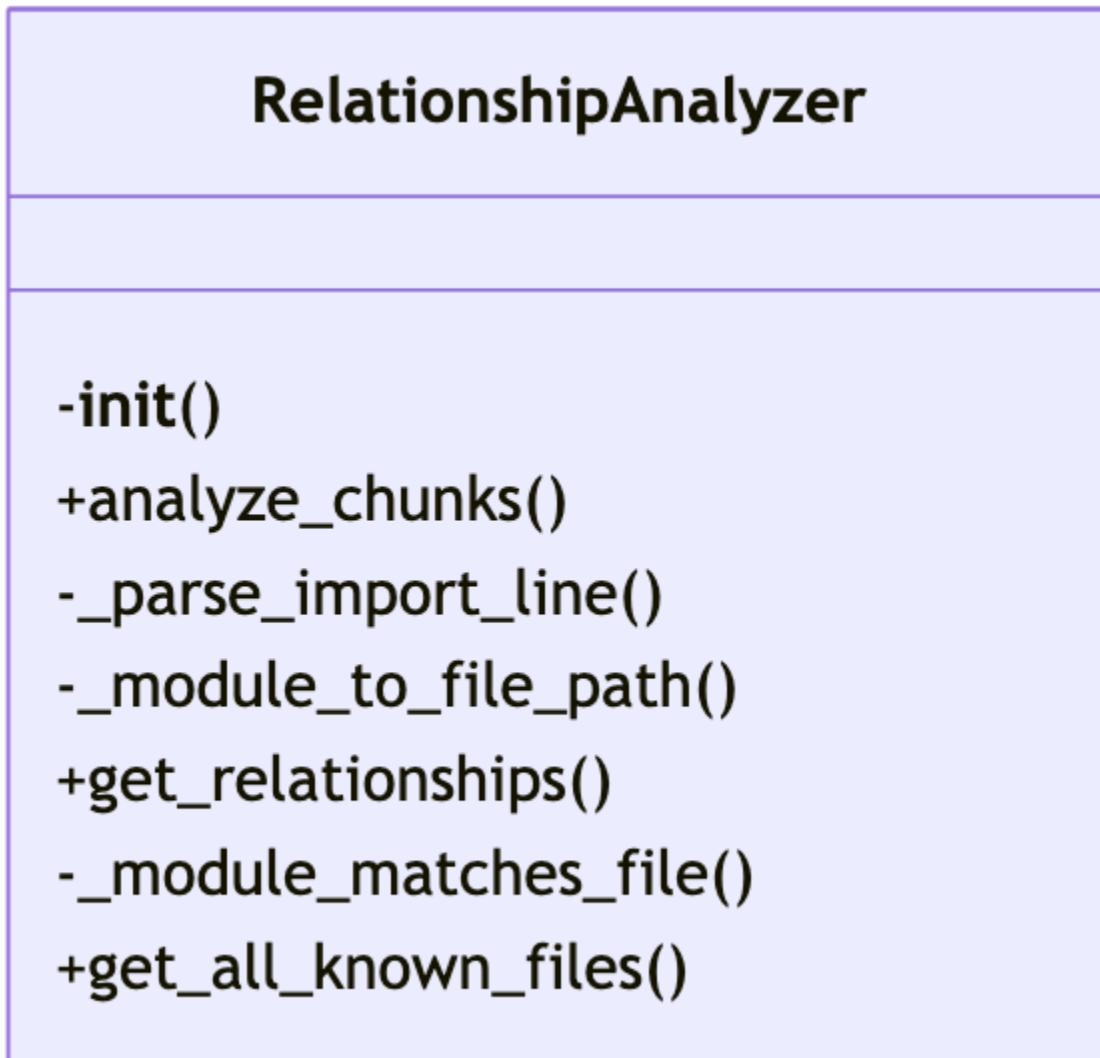
```
def add_see_also_sections(pages: list[WikiPage], analyzer: RelationshipAnalyzer) -> list[WikiPage]
```

Add See Also sections to wiki pages.

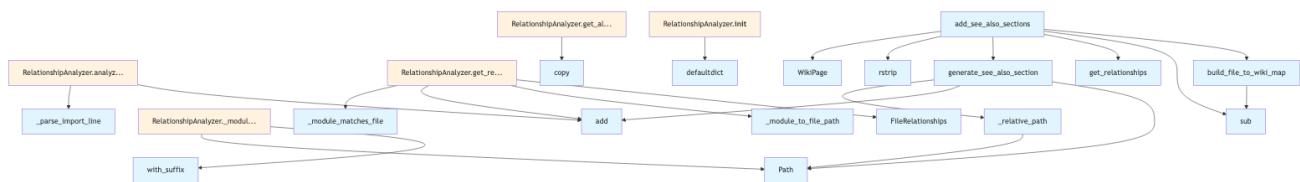
Parameter	Type	Default	Description
pages	list[WikiPage]	-	List of wiki pages.
analyzer	RelationshipAnalyzer	-	Relationship analyzer with import data.

Returns: list[WikiPage]

Class Diagram



Call Graph



Relevant Source Files

- `src/local_deepwiki/generators/see_also.py:16–22`

See Also

- [wiki](#) - uses this
- [crosslinks](#) - shares 4 dependencies
- [diagrams](#) - shares 4 dependencies
- [api_docs](#) - shares 4 dependencies

File Overview

This file defines data models and enumerations used in the local_deepwiki application. It provides the foundational data structures for representing language types, chunk types, file information, wiki pages, and search results. The models are built using Pydantic for data validation and serialization.

Classes

Language

An enumeration representing supported programming languages.

ChunkType

An enumeration representing different types of code chunks.

CodeChunk

A data model representing a code chunk with fields for content, language, and type.

Fields: - content (str): The content of the code chunk - language (Language): The programming language of the chunk - type (ChunkType): The type of the chunk

FileInfo

A data model representing file information.

Fields: - path (Path): The file path - size (int): The file size in bytes - modified_time (float): The last modified timestamp

IndexStatus

An enumeration representing the indexing status of a wiki page.

WikiPage

A data model representing a wiki page.

Fields: - title (str): The page title - content (str): The page content - file_info (FileInfo): Information about the source file - status (IndexStatus): The indexing status of the page - chunks (list[CodeChunk]): List of code chunks in the page

WikiStructure

A data model representing the overall structure of a wiki.

Fields: - pages (list[WikiPage]): List of wiki pages - title (str): The wiki title

SearchResult

A data model representing a search result.

Fields: - page (WikiPage): The wiki page containing the result - snippet (str): A snippet of text containing the search term - score (float): The relevance score of the result

WikiPageStatus

An enumeration representing the status of a wiki page.

WikiGenerationStatus

An enumeration representing the status of a wiki generation process.

Functions

No functions are defined in this file.

Usage Examples

```
from src.local_deepwiki.models import WikiPage, FileInfo, CodeChunk, Language, ChunkType

# Create a file info object
file_info = FileInfo(
    path=Path("example.py"),
    size=1024,
    modified_time=1634567890.0
)

# Create a code chunk
chunk = CodeChunk(
    content="print('Hello World')",
    language=Language.PYTHON,
    type=ChunkType.CODE
)

# Create a wiki page
page = WikiPage(
    title="Example Page",
    content="This is an example page",
    file_info=file_info,
    status=IndexStatus.INDEXED,
    chunks=[chunk]
)
```

Related Components

This file imports and uses: - `Enum` from `enum` module - `Path` from `pathlib` module - `Any` from `typing` module - `BaseModel` and `Field` from `pydantic` module

The models defined in this file are likely used by other components in the `local_deepwiki` application for data representation and validation.

API Reference

class Language

Inherits from: str , Enum

Supported programming languages.

class ChunkType

Inherits from: str , Enum

Types of code chunks.

class CodeChunk

Inherits from: BaseModel

A chunk of code extracted from the repository.

class FileInfo

Inherits from: BaseModel

Information about a source file.

class IndexStatus

Inherits from: BaseModel

Status of repository indexing.

class WikiPage

Inherits from: BaseModel

A generated wiki page.

class `WikiStructure`

Inherits from: `BaseModel`

Structure of the generated wiki.

Methods:

`to_toc`

```
def to_toc() -> dict[str, Any]
```

Generate table of contents.

class `SearchResult`

Inherits from: `BaseModel`

A search result from semantic search.

class `WikiPageStatus`

Inherits from: `BaseModel`

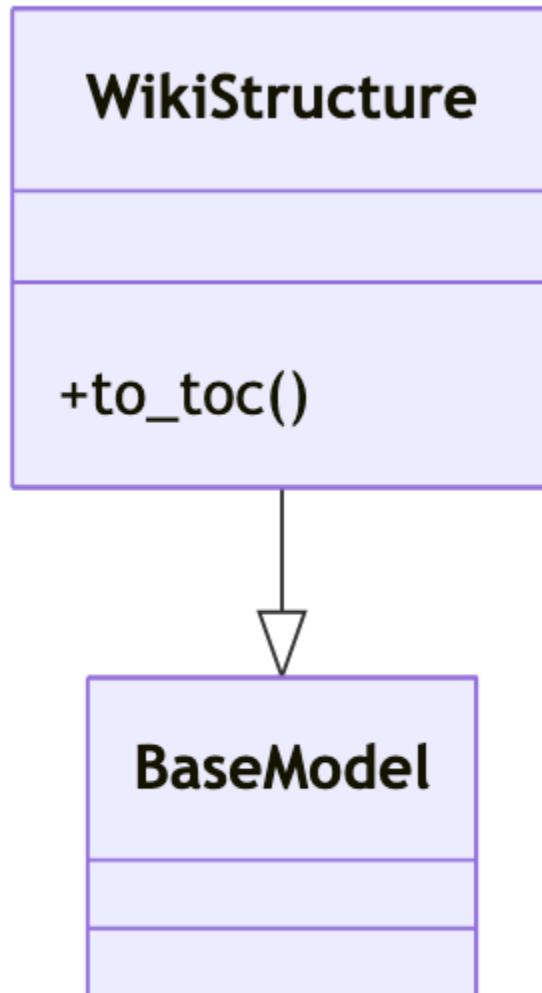
Status of a generated wiki page for incremental generation.

class `WikiGenerationStatus`

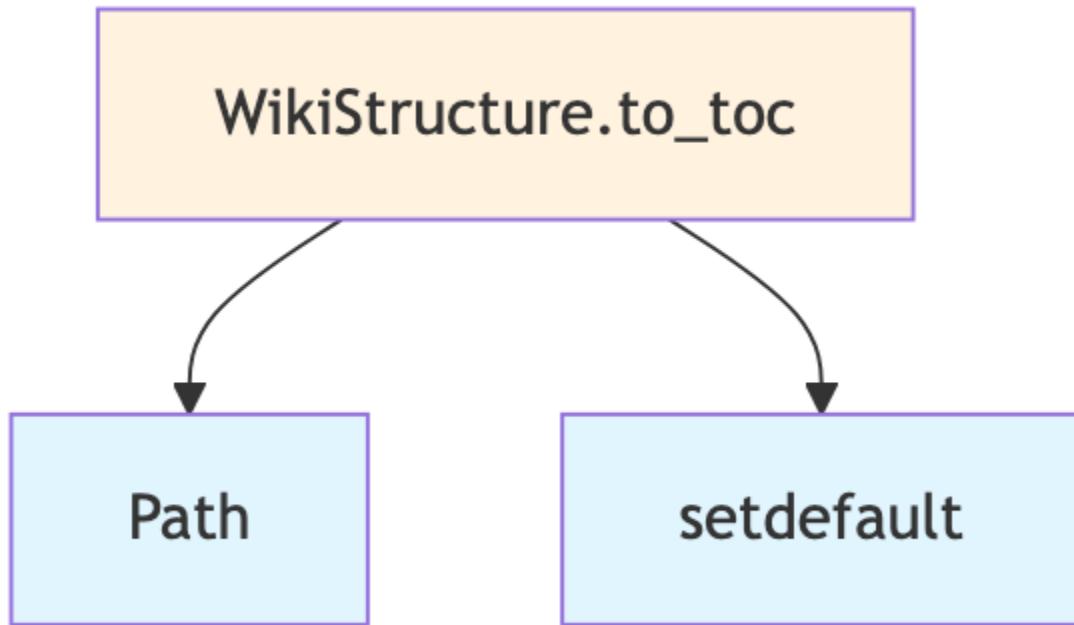
Inherits from: `BaseModel`

Status of wiki generation for tracking incremental updates.

Class Diagram



Call Graph



Relevant Source Files

- `src/local_deepwiki/models.py`

See Also

- [test_incremental_wiki](#) - uses this
- [api_docs](#) - uses this
- [test_see_also](#) - uses this
- [wiki](#) - uses this
- [test_chunker](#) - uses this

Base Provider Classes

File Overview

This file defines abstract base classes for embedding and language model providers. It establishes the interface contract that concrete provider implementations must follow, ensuring consistency across different embedding and LLM services.

Classes

EmbeddingProvider

Abstract base class for embedding providers. All concrete embedding implementations must inherit from this class and implement its abstract methods.

Methods

```
embed(texts: list[str]) -> list[list[float]]
```

Generate embeddings for a list of texts.

Parameters: - `texts` (`list[str]`): List of text strings to embed

Returns: - `list[list[float]]` : List of embedding vectors

Example:

```
# This method must be implemented by concrete classes
async def embed(self, texts: list[str]) -> list[list[float]]:
    # Implementation would generate embeddings for the provided texts
    pass
```

```
get_dimension() -> int
```

Get the embedding dimension.

Returns: - `int` : The dimension of the embedding vectors

Example:

```
# This method must be implemented by concrete classes
def get_dimension(self) -> int:
    # Implementation would return the dimension of embeddings
    pass
```

dimension (property)

Returns the embedding dimension as a property.

Returns: - `int` : The dimension of the embedding vectors

Dependencies

- `abc.ABC` : Abstract base class functionality
- `typing.AsyncIterator` : Type hint for asynchronous iterators

Usage Examples

To create a concrete embedding provider:

```
from base import EmbeddingProvider

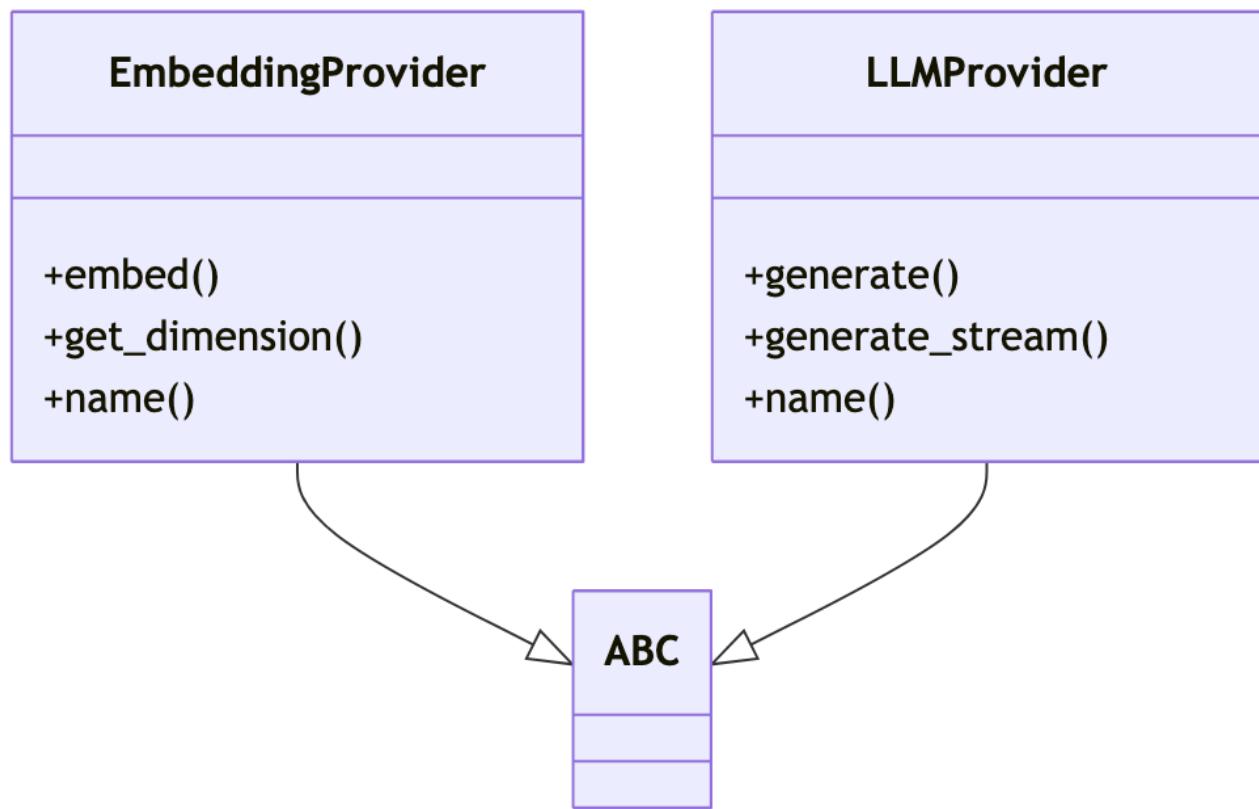
class MyEmbeddingProvider(EmbeddingProvider):
    def __init__(self):
        self._dimension = 768

    async def embed(self, texts: list[str]) -> list[list[float]]:
        # Implementation here
        pass

    def get_dimension(self) -> int:
        return self._dimension

# Usage
provider = MyEmbeddingProvider()
dimension = provider.get_dimension() # Returns 768
```

Class Diagram



See Also

- [vectorstore](#) - uses this
- [openai](#) - uses this
- [ollama](#) - uses this
- [openai](#) - uses this
- [local](#) - uses this

Local Embedding Provider

File Overview

This file implements a local embedding provider using the sentence-transformers library. It provides an implementation of the `EmbeddingProvider` base class that generates dense vector embeddings for text using pre-trained sentence transformers models.

Classes

LocalEmbeddingProvider

The `LocalEmbeddingProvider` class implements the `EmbeddingProvider` interface to generate text embeddings using local sentence transformer models.

Key Methods:

- `__init__(self, model_name: str = "all-MiniLM-L6-v2")` : Initializes the provider with a specified model name
- `get_embedding(self, text: str) -> list[float]` : Generates a vector embedding for the given text
- `get_embeddings(self, texts: list[str]) -> list[list[float]]` : Generates vector embeddings for multiple texts

Usage:

```
from local_deepwiki.providers.embeddings.local import LocalEmbeddingProvider

# Initialize provider
provider = LocalEmbeddingProvider(model_name="all-MiniLM-L6-v2")

# Generate single embedding
embedding = provider.get_embedding("Hello world")

# Generate multiple embeddings
embeddings = provider.get_embeddings(["Hello world", "How are you?"])
```

Functions

```
__init__(self, model_name: str = "all-MiniLM-L6-v2")
```

Initializes the LocalEmbeddingProvider with a specified sentence transformer model.

Parameters: - `model_name` (str): Name of the sentence transformer model to use. Defaults to "all-MiniLM-L6-v2"

Returns: - None

```
get_embedding(self, text: str) -> list[float]
```

Generates a vector embedding for a single text string.

Parameters: - `text` (str): The text to embed

Returns: - `list[float]` : The embedding vector as a list of floating point numbers

```
get_embeddings(self, texts: list[str]) -> list[list[float]]
```

Generates vector embeddings for multiple text strings.

Parameters: - `texts` (list[str]): List of texts to embed

Returns: - `list[list[float]]` : List of embedding vectors, one for each input text

Usage Examples

```
# Basic usage
from local_deepwiki.providers.embeddings.local import LocalEmbeddingProvider

# Create provider with default model
provider = LocalEmbeddingProvider()

# Generate single embedding
text = "The quick brown fox jumps over the lazy dog"
embedding = provider.get_embedding(text)
print(f"Embedding shape: {len(embedding)}")

# Generate multiple embeddings
texts = [
    "Hello world",
    "How are you?",
    "I am fine, thank you"
]
embeddings = provider.get_embeddings(texts)
print(f"Generated {len(embeddings)} embeddings")
```

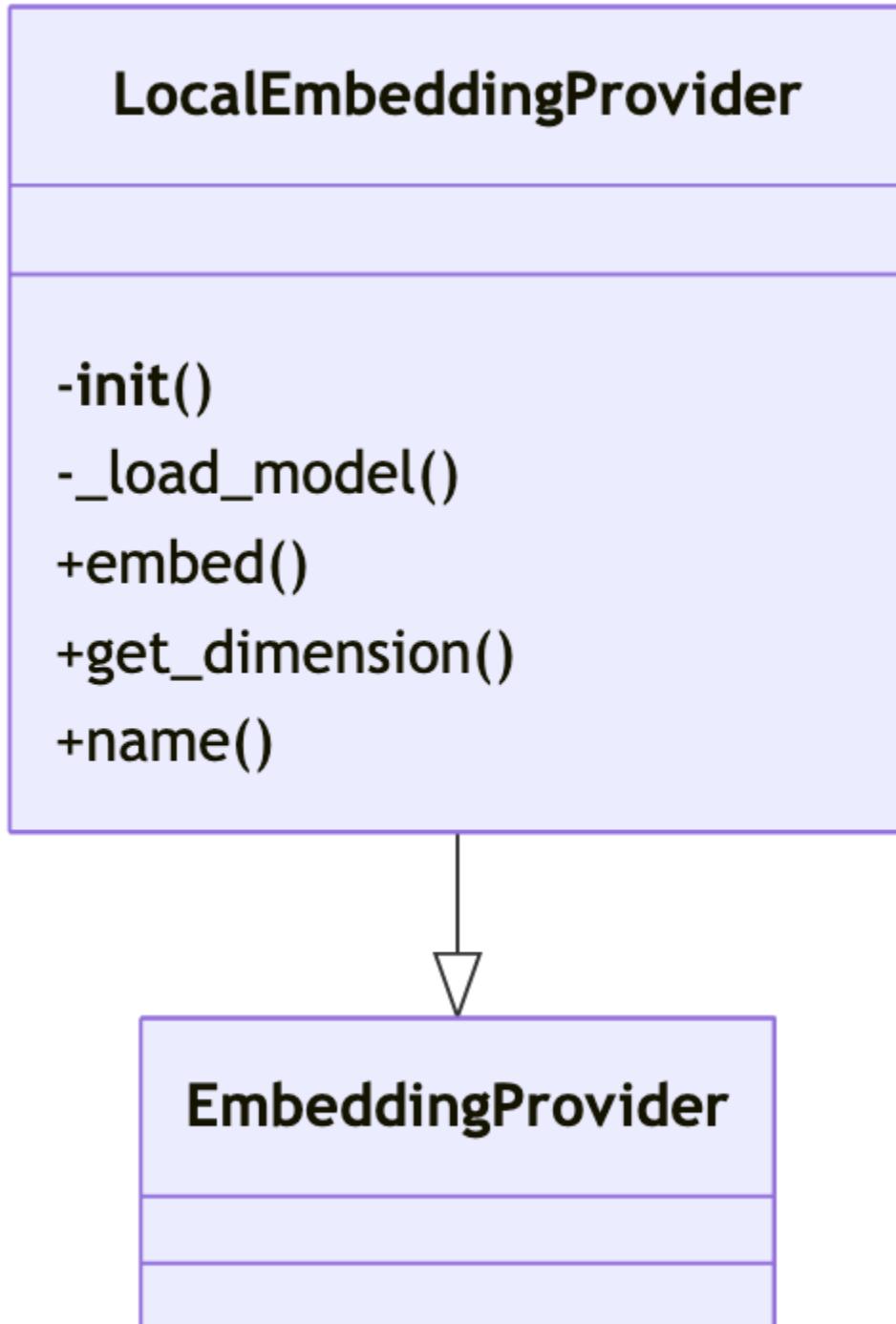
Dependencies

This file depends on: - `sentence_transformers.SentenceTransformer` : For generating sentence embeddings - `local_deepwiki.providers.base.EmbeddingProvider` : Base class interface for embedding providers

The package requires the `sentence-transformers` library to be installed:

```
pip install sentence-transformers
```

Class Diagram



See Also

- [base](#) - dependency

OpenAI Embedding Provider

File Overview

This file implements an embedding provider that uses the OpenAI API to generate embeddings for text. It provides an asynchronous interface for generating embeddings using OpenAI's embedding models.

Classes

`OpenAIEmbeddingProvider`

An embedding provider that uses OpenAI's API to generate text embeddings.

Constructor

```
def __init__(self, model: str = "text-embedding-3-small", api_key: str | None = None)
```

Initialize the OpenAI embedding provider.

Parameters: - `model` (str): OpenAI embedding model name. Defaults to "text-embedding-3-small" - `api_key` (str | None): Optional API key. Uses OPENAI_API_KEY environment variable if not provided

Attributes: - `_model` (str): The embedding model name - `_client` (AsyncOpenAI): Async OpenAI client instance - `_dimension` (int): Embedding dimension size for the specified model

Methods

```
async def embed(self, texts: list[str]) -> list[list[float]]
```

Generate embeddings for a list of texts.

Parameters: - `texts` (list[str]): List of text strings to embed

Returns: - `list[list[float]]` : List of embedding vectors, where each vector is a list of floats

Usage Examples

Basic Usage

```
from local_deepwiki.providers.embeddings.openai import OpenAIEEmbeddingProvider

# Using default model and API key from environment
provider = OpenAIEEmbeddingProvider()

# Using custom model and API key
provider = OpenAIEEmbeddingProvider(
    model="text-embedding-3-large",
    api_key="your-api-key-here"
)
```

Generating Embeddings

```
import asyncio

async def example():
    provider = OpenAIEEmbeddingProvider()

    texts = [
        "Hello world",
        "How are you?",
        "OpenAI embeddings"
    ]

    embeddings = await provider.embed(texts)
    print(embeddings)

# Run the async function
asyncio.run(example())
```

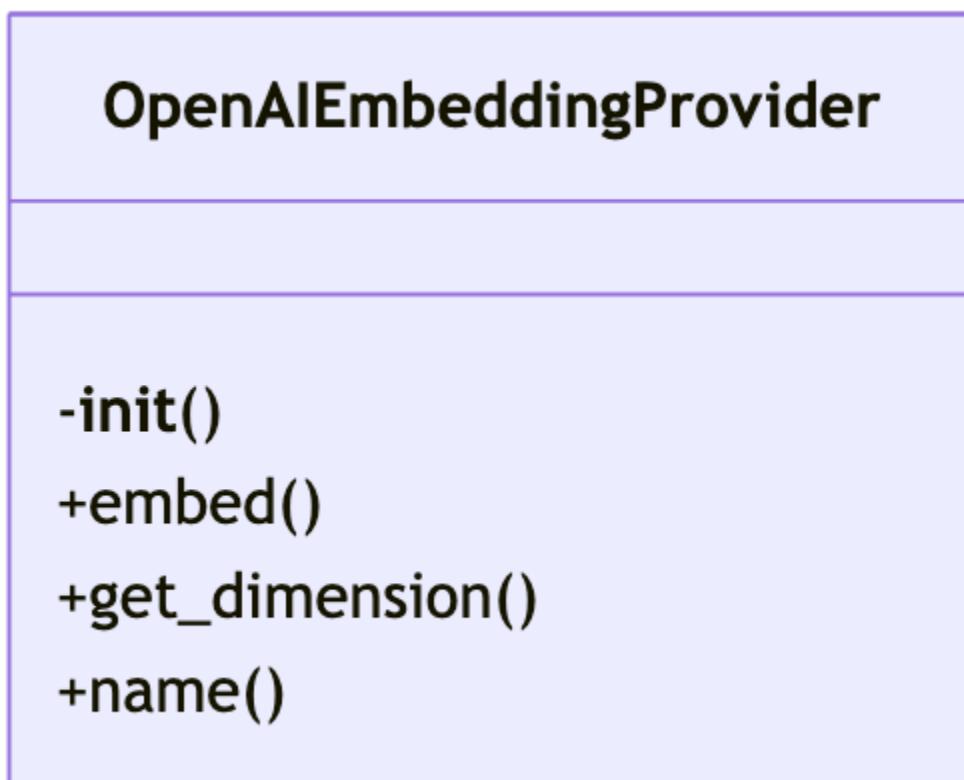
Dependencies

- `os` : For accessing environment variables

- `openai.AsyncOpenAI` : OpenAI asynchronous client
- `local_deepwiki.providers.base.EmbeddingProvider` : Base embedding provider class

The provider requires the `OPENAI_API_KEY` environment variable to be set if no explicit API key is provided during initialization.

Class Diagram



See Also

- [base](#) - dependency
- [openai](#) - shares 3 dependencies
- [anthropic](#) - shares 2 dependencies

Anthropic Provider Documentation

File Overview

This file implements the Anthropic LLM provider for the local_deepwiki system. It provides an asynchronous interface to interact with Anthropic's Claude models through their API, extending the base LLMProvider class to integrate with the system's provider architecture.

Classes

`AnthropicProvider`

Purpose: An asynchronous provider that implements the LLMProvider interface for Anthropic's Claude models. This class handles authentication, model selection, and asynchronous generation of responses from Anthropic's API.

Key Methods:

- `__init__(self, model: str = "claude-3-haiku-20240307")` : Initializes the provider with a default model name
- `generate(self, prompt: str) -> AsyncIterator[str]` : Asynchronously generates text responses from the LLM
- `get_model_name(self) -> str` : Returns the name of the configured model

Usage:

```
provider = AnthropicProvider(model="claude-3-sonnet-20240229")
async for chunk in provider.generate("Hello, world!"):
    print(chunk)
```

Functions

```
__init__(self, model: str = "claude-3-haiku-20240307")
```

Parameters: - `model` (str): The name of the Anthropic model to use (default: "claude-3-haiku-20240307")

Purpose: Initializes the Anthropic provider with the specified model and sets up the `AsyncAnthropic` client using the API key from the environment.

```
generate(self, prompt: str) -> AsyncIterator[str]
```

Parameters: - `prompt` (str): The input prompt to send to the LLM

Return Value: - `AsyncIterator[str]` : An asynchronous iterator that yields response chunks from the LLM

Purpose: Generates text responses from the configured Anthropic model asynchronously, yielding response chunks as they become available.

```
get_model_name(self) -> str
```

Return Value: - `str` : The name of the configured Anthropic model

Purpose: Returns the name of the model currently configured for use with this provider.

Usage Examples

Basic Usage

```
from local_deepwiki.providers.llm.anthropic import AnthropicProvider

# Initialize provider with default model
provider = AnthropicProvider()

# Generate response
async for chunk in provider.generate("What is the capital of France?"):
    print(chunk)
```

Custom Model Selection

```
from local_deepwiki.providers.llm.anthropic import AnthropicProvider

# Initialize provider with specific model
provider = AnthropicProvider(model="claude-3-sonnet-20240229")

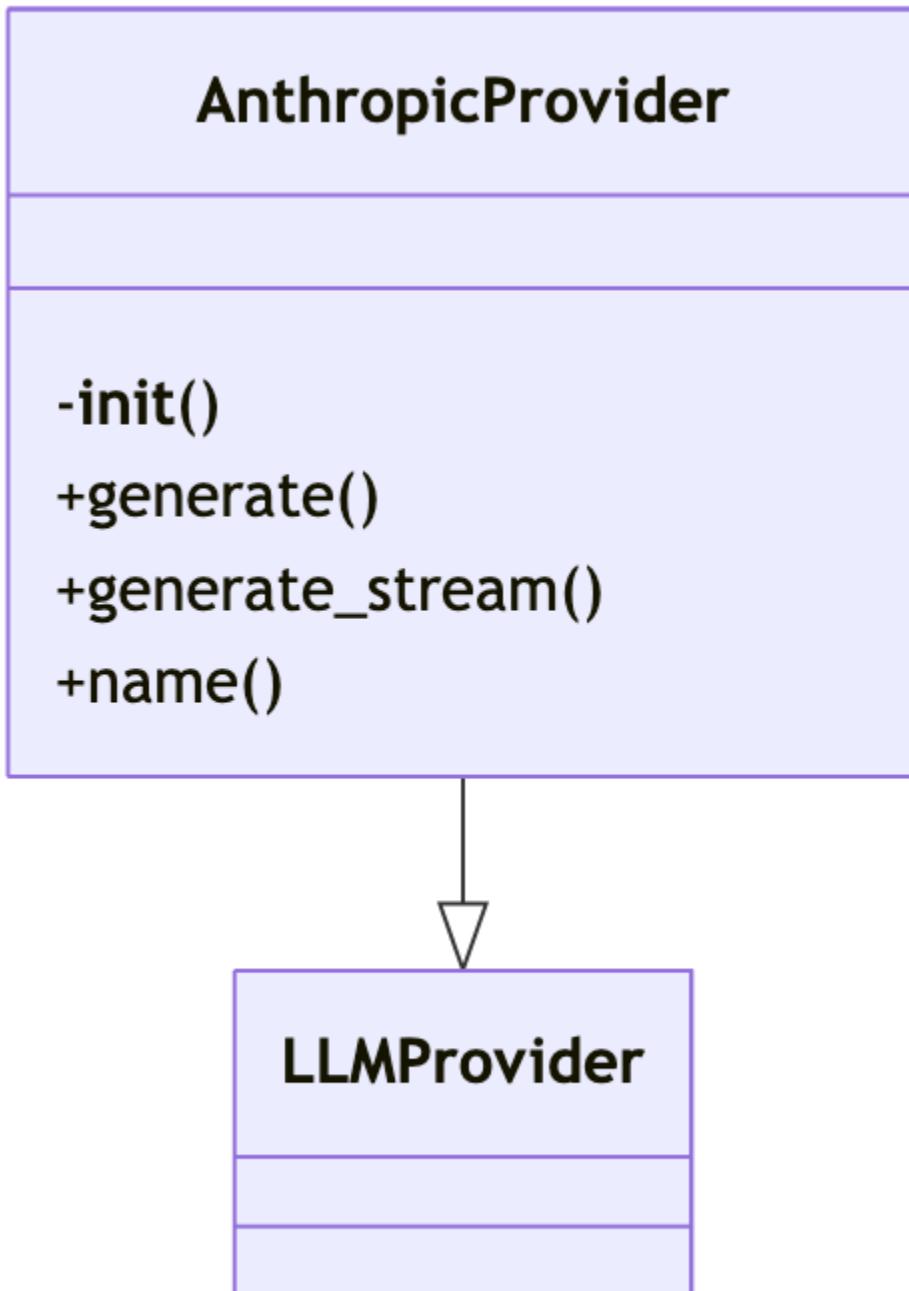
# Get model name
model_name = provider.get_model_name()
print(f"Using model: {model_name}")
```

Dependencies

This file depends on: - `os` : For accessing environment variables (specifically `ANTHROPIC_API_KEY`) - `typing.AsyncIterator` : For type hints of asynchronous iterators - `anthropic.AsyncAnthropic` : The official Anthropic SDK for asynchronous API calls - `local_deepwiki.providers.base.LLMProvider` : Base class that this provider implements

Environment Requirements: - `ANTHROPIC_API_KEY` must be set in the environment variables for API authentication

Class Diagram



See Also

- [base](#) - dependency
- [openai](#) - shares 3 dependencies
- [config](#) - shares 2 dependencies
- [vectorstore](#) - shares 2 dependencies

Ollama Provider Documentation

File Overview

This file implements an LLM provider that interfaces with local Ollama instances. It provides an asynchronous interface for generating text using local large language models through the Ollama API.

Classes

OllamaProvider

Purpose: An asynchronous LLM provider that communicates with a local Ollama instance to generate text responses.

Key Methods:

- `__init__(model: str = "llama3.2", base_url: str = "http://localhost:11434")`
- `generate(prompt: str, system_prompt: str | None = None, max_tokens: int = 4096, temperature: float = 0.7, top_p: float = 0.9) -> AsyncIterator[str]`

Usage:

```
provider = OllamaProvider(model="llama3.2", base_url="http://localhost:11434")
async for chunk in provider.generate("Hello, world!"):
    print(chunk)
```

Functions

generate

Purpose: Asynchronously generates text using the configured Ollama model.

Parameters: - `prompt` (str): The user prompt to send to the LLM - `system_prompt` (str | None, optional): System prompt to prepend to the conversation. Defaults to `None` - `max_tokens` (int, optional): Maximum number of tokens to generate. Defaults to `4096` - `temperature` (float, optional): Sampling temperature for response randomness. Defaults to `0.7` - `top_p` (float, optional): Nucleus sampling parameter. Defaults to `0.9`

Return Value: - `AsyncIterator[str]` : An asynchronous iterator yielding text chunks as they are generated

Usage:

```
async for chunk in provider.generate("What is AI?", max_tokens=1000, temperature=0.5):
    print(chunk)
```

Usage Examples

Basic Usage

```
from local_deepwiki.providers.llm.ollama import OllamaProvider

# Initialize provider
provider = OllamaProvider()

# Generate text
async for chunk in provider.generate("Write a poem about technology"):
    print(chunk)
```

Custom Configuration

```
from local_deepwiki.providers.llm.ollama import OllamaProvider

# Initialize with custom model and URL
provider = OllamaProvider(
    model="mistral",
    base_url="http://192.168.1.100:11434"
)

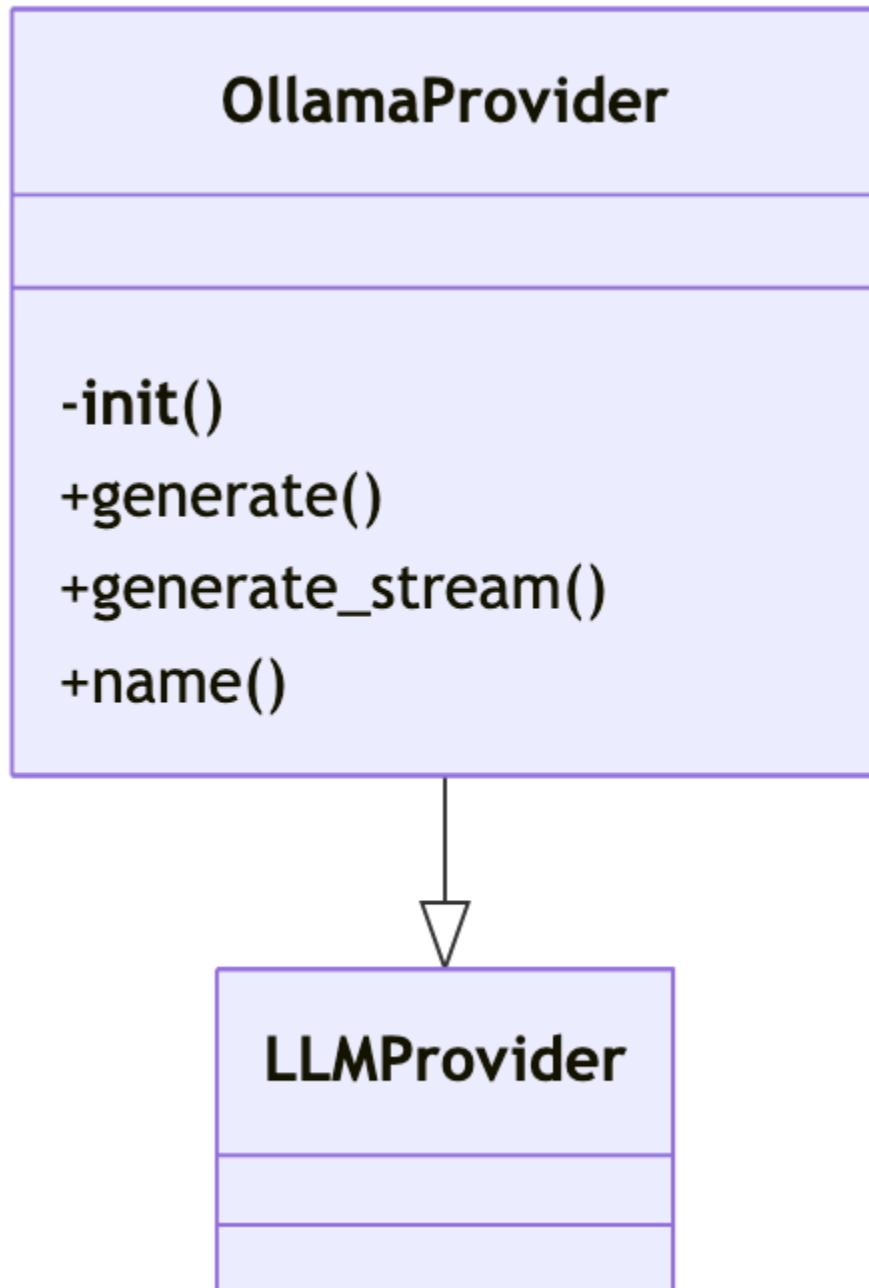
# Generate with custom parameters
async for chunk in provider.generate(
    prompt="Explain quantum computing",
    max_tokens=500,
    temperature=0.3,
    top_p=0.8
):
    print(chunk)
```

Dependencies

This file imports: - `AsyncIterator` from `typing` - For type hinting asynchronous iterators - `AsyncClient` from `ollama` - For asynchronous communication with Ollama API - `LLMProvider` from `local_deepwiki.providers.base` - Base class for LLM providers

The provider requires a running Ollama instance accessible at the specified `base_url` and the requested `model` to be available locally.

Class Diagram



See Also

- [base](#) - dependency
- [vectorstore](#) - shares 2 dependencies
- [anthropic](#) - shares 2 dependencies
- [openai](#) - shares 2 dependencies

OpenAI LLM Provider

File Overview

This file implements an OpenAI LLM provider that extends the base LLM provider interface. It provides asynchronous access to OpenAI's language models through the `AsyncOpenAI` client, enabling text generation with configurable models and API keys.

Classes

`OpenAILLMProvider`

Purpose: An asynchronous LLM provider that interfaces with OpenAI's API to generate text responses.

Key Methods:

- `__init__(model: str = "gpt-4o", api_key: str | None = None)` : Initializes the provider with a model name and optional API key
- `generate(prompt: str, system_prompt: str | None = None, max_tokens: int = 4096, temperature: float = 0.7) -> AsyncIterator[str]` : Asynchronously generates text from a prompt

Usage:

```
provider = OpenAILLMProvider(model="gpt-4-turbo", api_key="your-api-key")
async for chunk in provider.generate("Hello, world!"):
    print(chunk)
```

Functions

```
__init__(self, model: str = "gpt-4o", api_key: str | None = None)
```

Initializes the OpenAI LLM provider.

Parameters: - `model` (str): The OpenAI model to use (default: "gpt-4o") - `api_key` (str | None): Optional API key. If not provided, uses the `OPENAI_API_KEY` environment variable

Returns: None

```
generate(self, prompt: str, system_prompt: str | None = None = None, max_tokens: int = 4096,
temperature: float = 0.7) -> AsyncIterator[str]
```

Asynchronously generates text from a given prompt.

Parameters: - `prompt` (str): The user prompt to generate text from - `system_prompt` (str | None): Optional system prompt to guide the model's behavior - `max_tokens` (int): Maximum number of tokens to generate (default: 4096) - `temperature` (float): Sampling temperature (default: 0.7)

Returns: `AsyncIterator[str]` - An asynchronous iterator yielding text chunks as they are generated

Usage Examples

Basic Usage

```
from local_deepwiki.providers.llm.openai import OpenAILLMProvider

# Initialize provider (uses OPENAI_API_KEY from environment)
provider = OpenAILLMProvider()

# Generate text
async for chunk in provider.generate("Explain quantum computing in simple terms"):
    print(chunk)
```

With Custom Model and API Key

```
provider = OpenAILLMProvider(  
    model="gpt-4-turbo",  
    api_key="sk-your-api-key-here"  
)  
  
async for chunk in provider.generate(  
    prompt="Write a poem about technology",  
    max_tokens=500,  
    temperature=0.8  
):  
    print(chunk)
```

With System Prompt

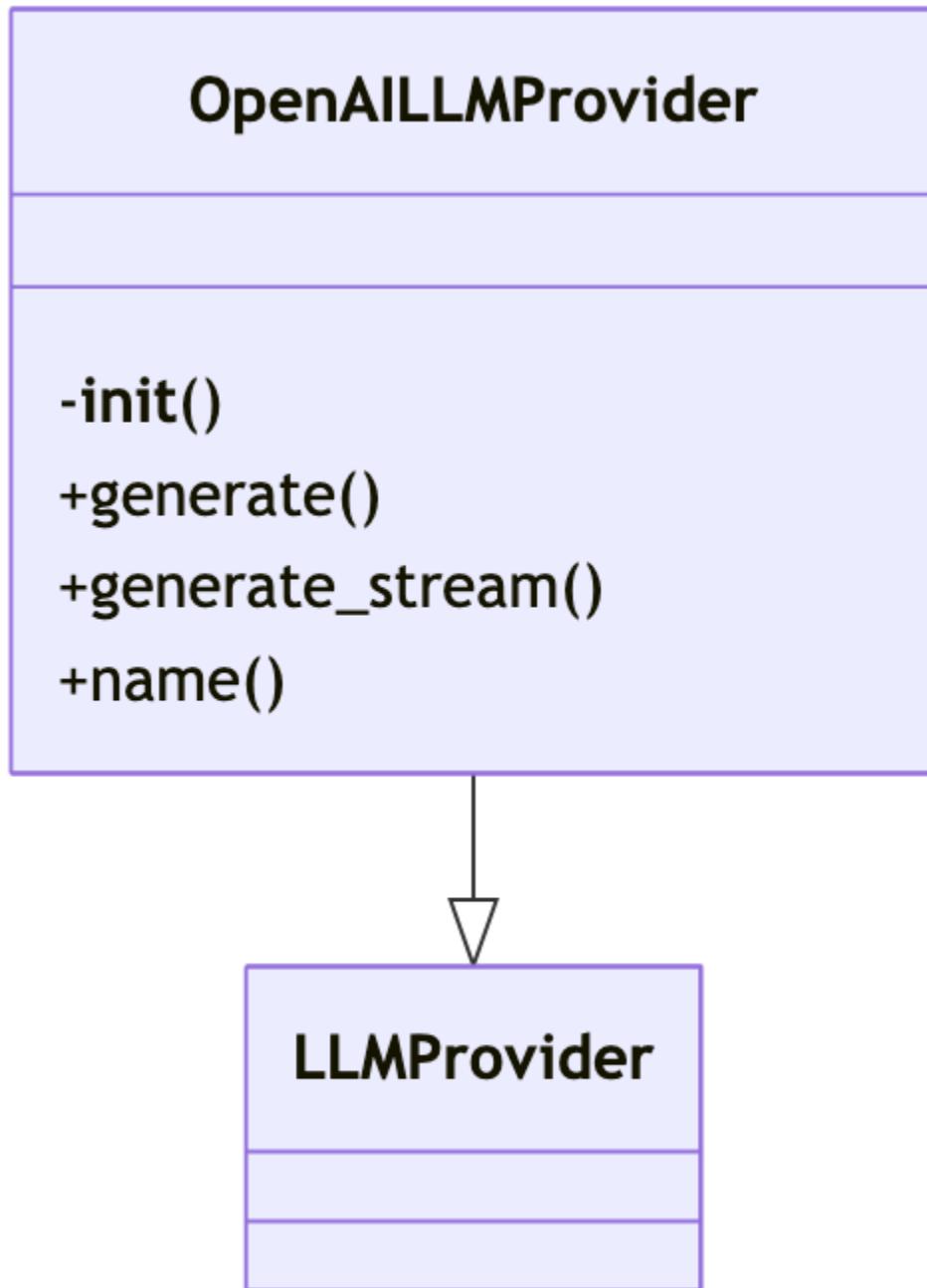
```
async for chunk in provider.generate(  
    prompt="What is the capital of France?",  
    system_prompt="You are a helpful assistant that provides concise answers."  
):  
    print(chunk)
```

Dependencies

This file imports: - `os` - For accessing environment variables - `AsyncIterator` - From `typing` - Type hint for asynchronous iterators - `AsyncOpenAI` - From `openai` - OpenAI async client - `LLMProvider` - From `local_deepwiki.providers.base` - Base provider interface

The provider requires the `openai` Python package to be installed and the `OPENAI_API_KEY` environment variable to be set for basic functionality.

Class Diagram



See Also

- [openai](#) - dependency
- [base](#) - dependency
- [anthropic](#) - shares 3 dependencies
- [config](#) - shares 2 dependencies

File Overview

This file, `src/local_deepwiki/web/app.py`, serves as the main entry point for the DeepWiki web server. It provides functionality to create and run a Flask-based web application that serves documentation from a specified wiki directory. The application supports rendering markdown files as HTML, building navigation breadcrumbs, and searching through wiki content.

Functions

`run_server`

```
def run_server(wiki_path: str | Path, host: str = "127.0.0.1", port: int = 8080, debug: bool = False)
```

Run the wiki web server.

Parameters: - `wiki_path` (str | Path): Path to the wiki directory. - `host` (str): Host address to bind the server to. Defaults to `"127.0.0.1"`. - `port` (int): Port number to bind the server to. Defaults to `8080`. - `debug` (bool): Enable debug mode. Defaults to `False`.

`create_app`

```
def create_app(wiki_path: str | Path) -> Flask:
```

Create Flask app with wiki path configured.

Parameters: - `wiki_path` (str | Path): Path to the wiki directory.

Returns: - `Flask`: A configured Flask application instance.

main

```
def main():
```

CLI entry point.

Parses command-line arguments and starts the web server using the provided arguments.

Usage Examples

To start the DeepWiki server from the command line:

```
python src/local_deepwiki/web/app.py /path/to/wiki --host 0.0.0.0 --port 8080 --debug
```

Or with default settings (serving from `.deepwiki` directory):

```
python src/local_deepwiki/web/app.py
```

Related Components

This file imports and uses components from the following modules: - `flask` : For creating the web application and handling HTTP requests. - `markdown` : For rendering markdown content into HTML. - `argparse` : For parsing command-line arguments. - `pathlib.Path` : For handling file paths.

The file also references the global variable `WIKI_PATH` , which is set during the `create_app` function call. It depends on the structure of a wiki directory that contains markdown files to be rendered.

API Reference

Functions

get_wiki_structure

```
def get_wiki_structure(wiki_path: Path) -> tuple[list, dict, list | None]
```

Get wiki pages and sections, with optional hierarchical TOC.

Parameter	Type	Default	Description
wiki_path	Path	-	-

Returns: tuple[list, dict, list | None]

extract_title

```
def extract_title(md_file: Path) -> str
```

Extract title from markdown file.

Parameter	Type	Default	Description
md_file	Path	-	-

Returns: str

render_markdown

```
def render_markdown(content: str) -> str
```

Render markdown to HTML.

Parameter	Type	Default	Description
content	str	-	-

Returns: str**build_breadcrumb**

```
def buildBreadcrumb(wiki_path: Path, current_path: str) -> str
```

Build breadcrumb navigation HTML with clickable links. For a path like 'files/src/local_deepwiki/core/chunker.md', generates: Home > Files > src > local_deepwiki > core > chunker Each segment links to its index.md if one exists in that folder.

Parameter	Type	Default	Description
wiki_path	Path	-	-
current_path	str	-	-

Returns: str**index**

```
@app.route('/')
```

```
def index()
```

Redirect to index.md.

search_json

```
@app.route('/search.json')
```

```
def search_json()
```

Serve the search index JSON file.

view_page

```
@app.route('/wiki/<path:path>')
```

```
def view_page(path: str)
```

View a wiki page.

Parameter	Type	Default	Description
path	str	-	-

create_app

```
def create_app(wiki_path: str | Path) -> Flask
```

Create Flask app with wiki path configured.

Parameter	Type	Default	Description
wiki_path	str Path	-	-

Returns: Flask

run_server

```
def run_server(wiki_path: str | Path, host: str = "127.0.0.1", port: int = 8080, debug: bool = False)
```

Run the wiki web server.

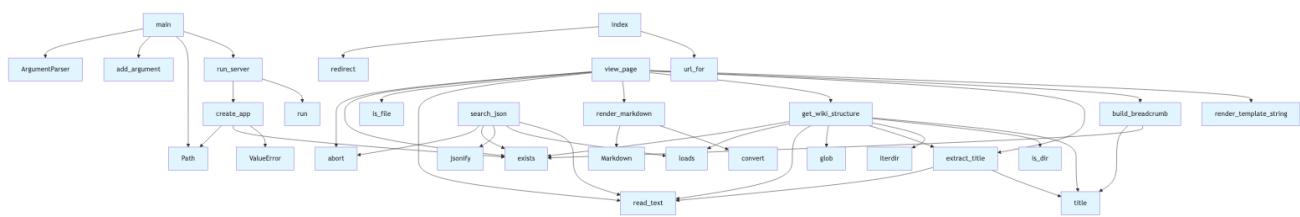
Parameter	Type	Default	Description
wiki_path	str Path	-	-
host	str	"127.0.0.1"	-
port	int	8080	-
debug	bool	False	-

main

```
def main()
```

CLI entry point.

Call Graph



Relevant Source Files

- `src/local_deepwiki/web/app.py`

File Overview

This file contains test cases for the call graph extraction functionality in the `local_deepwiki` library. It tests various aspects of extracting function calls from code and generating call graphs, including support for Python, JavaScript, and Go languages.

Classes

TestCallGraphExtractor

Test the CallGraphExtractor class.

Methods

- `extractor` (fixture): Returns a new instance of CallGraphExtractor
- `test_extract_from_simple_file` : Test extracting call graph from a simple Python file

TestGetFileCallGraph

Test the convenience function for getting file call graph.

Methods

- `test_returns_diagram_for_file_with_calls` : Test that diagram is returned for file with function calls

TestGenerateCallGraphDiagram

Test Mermaid diagram generation.

Methods

- `test_empty_graph_returns_none` : Test that empty call graph returns None
- `test_simple_diagram` : Test generating a simple diagram

TestExtractCallsPython

Test extraction of function calls from Python code.

Methods

- `parser` (fixture): Returns a new instance of [CodeParser](#)
- `test_simple_function_call` : Test extracting a simple function call
- `test_multiple_function_calls` : Test extracting multiple function calls
- `test_method_call` : Test extracting method calls
- `test_nested_calls` : Test extracting nested function calls
- `test_filters_builtin` : Test that built-ins are filtered out
- `test_deduplicates_calls` : Test that duplicate calls are removed

TestGoCallExtraction

Test call extraction for Go code.

Methods

- `parser` (fixture): Returns a new instance of [CodeParser](#)
- `test_simple_go_call` : Test extracting calls from Go function

TestJavaScriptCallExtraction

Test call extraction for JavaScript code.

Methods

- `parser` (fixture): Returns a new instance of [CodeParser](#)
- `test_simple_js_call` : Test extracting calls from JavaScript function

Functions

`_is_builtin_or_noise`

Filters out built-in functions and noise from call graphs.

`extract_call_name`

Extracts the name of a function call from an AST node.

`extract_calls_from_function`

Extracts all function calls from a function definition.

Parameters

- `func_node` : The AST node representing the function definition
- `source_bytes` : The source code as bytes
- `language` : The programming language (e.g., Language.PYTHON)

Returns

A list of function call names

`generate_call_graph_diagram`

Generates a Mermaid flowchart diagram from a call graph.

Parameters

- `call_graph` : A dictionary mapping function names to lists of called functions

Returns

A string containing the Mermaid diagram or None if the graph is empty

get_file_call_graph

Gets the call graph diagram for a given file.

Parameters

- `file_path` : Path to the source file
- `output_dir` : Directory to store the output

Returns

The generated call graph diagram as a string

Usage Examples

Extracting Calls from Python Code

```
from local_deepwiki.generators.callgraph import extract_calls_from_function
from local_deepwiki.core.parser import CodeParser
from local_deepwiki.models import Language

parser = CodeParser()
source = """
def main():
    process_data()
"""

root = parser.parse_source(source, Language.PYTHON)
func_node = root.children[0]

calls = extract_calls_from_function(func_node, source.encode(), Language.PYTHON)
# Returns ['process_data']
```

Generating a Call Graph Diagram

```
from local_deepwiki.generators.callgraph import generate_call_graph_diagram

call_graph = {
    "main": ["helper", "process"],
}

diagram = generate_call_graph_diagram(call_graph)
# Returns a Mermaid diagram string
```

Getting File Call Graph

```
from local_deepwiki.generators.callgraph import get_file_call_graph

diagram = get_file_call_graph("path/to/file.py", "output_dir")
# Returns the generated diagram as a string
```

Related Components

This file works with the following components:

- `CodeParser` from `local_deepwiki.core.parser` : Used for parsing source code into ASTs
- `Language` from `local_deepwiki.models` : Enum representing supported programming languages
- `find_nodes_by_type` from `local_deepwiki.core.parser` : Helper function for finding AST nodes by type
- `CallGraphExtractor` from `local_deepwiki.generators.callgraph` : Main class for extracting call graphs
- `extract_call_name` from `local_deepwiki.generators.callgraph` : Helper for extracting function call names
- `generate_call_graph_diagram` from `local_deepwiki.generators.callgraph` : Function for creating Mermaid diagrams
- `get_file_call_graph` from `local_deepwiki.generators.callgraph` : Convenience function for getting file call graphs

API Reference

`class TestIsBuiltinOrNoise`

Test filtering of built-in functions.

Methods:

test_common_builtins_filtered

```
def test_common_builtins_filtered()
```

Test that common built-ins are filtered.

test_python_specific_builtin

```
def test_python_specific_builtin()
```

Test Python-specific built-ins are filtered.

test_custom_functions_not_filtered

```
def test_custom_functions_not_filtered()
```

Test that custom function names are not filtered.

class TestExtractCallsPython

Test call extraction for Python code.

Methods:**parser**

```
def parser()
```

test_simple_function_call

```
def test_simple_function_call(parser)
```

Test extracting a simple function call.

Parameter	Type	Default	Description
parser	-	-	-

test_multiple_function_calls

```
def test_multiple_function_calls(parser)
```

Test extracting multiple function calls.

Parameter	Type	Default	Description
parser	-	-	-

test_method_call

```
def test_method_call(parser)
```

Test extracting method calls.

Parameter	Type	Default	Description
parser	-	-	-

test_nested_calls

```
def test_nested_calls(parser)
```

Test extracting nested function calls.

Parameter	Type	Default	Description
parser	-	-	-

test_filters_builtins

```
def test_filters_builtin(parser)
```

Test that built-ins are filtered out.

Parameter	Type	Default	Description
parser	-	-	-

test_deduplicates_calls

```
def test_deduplicates_calls(parser)
```

Test that duplicate calls are removed.

Parameter	Type	Default	Description
parser	-	-	-

class TestCallGraphExtractor

Test the CallGraphExtractor class.

Methods:**extractor**

```
def extractor()
```

test_extract_from_simple_file

```
def test_extract_from_simple_file(tmp_path, extractor)
```

Test extracting call graph from a simple Python file.

Parameter	Type	Default	Description
tmp_path	-	-	-
extractor	-	-	-

test_extract_class_methods

```
def test_extract_class_methods(tmp_path, extractor)
```

Test extracting call graph with class methods.

Parameter	Type	Default	Description
tmp_path	-	-	-
extractor	-	-	-

test_extract_mixed_functions_and_methods

```
def test_extract_mixed_functions_and_methods(tmp_path, extractor)
```

Test extracting from file with both functions and class methods.

Parameter	Type	Default	Description
tmp_path	-	-	-
extractor	-	-	-

class TestGenerateCallGraphDiagram

Test Mermaid diagram generation.

Methods:**test_empty_graph_returns_none**

```
def test_empty_graph_returns_none()
```

Test that empty call graph returns None.

test_simple_diagram

```
def test_simple_diagram()
```

Test generating a simple diagram.

test_diagram_with_methods

```
def test_diagram_with_methods()
```

Test diagram distinguishes functions from methods.

test_limits_nodes

```
def test_limits_nodes()
```

Test that diagram limits number of nodes.

test_sanitizes_long_names

```
def test_sanitizes_long_names()
```

Test that long names are truncated.

class TestGetFileCallGraph

Test the convenience function for getting file call graph.

Methods:**test_returns_diagram_for_file_with_calls**

```
def test_returns_diagram_for_file_with_calls(tmp_path)
```

Test that diagram is returned for file with function calls.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_returns_none_for_file_without_calls

```
def test_returns_none_for_file_without_calls(tmp_path)
```

Test that None is returned for file without function calls.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_returns_none_for_unsupported_file

```
def test_returns_none_for_unsupported_file(tmp_path)
```

Test that None is returned for unsupported file types.

Parameter	Type	Default	Description
tmp_path	-	-	-

class TestJavaScriptCallExtraction

Test call extraction for JavaScript code.

Methods:**parser**

```
def parser()
```

test_simple_js_call

```
def test_simple_js_call(parser)
```

Test extracting calls from JavaScript function.

Parameter	Type	Default	Description
parser	-	-	-

test_js_method_call

```
def test_js_method_call(parser)
```

Test extracting method calls in JavaScript.

Parameter	Type	Default	Description
parser	-	-	-

class TestGoCallExtraction

Test call extraction for Go code.

Methods:**parser**

```
def parser()
```

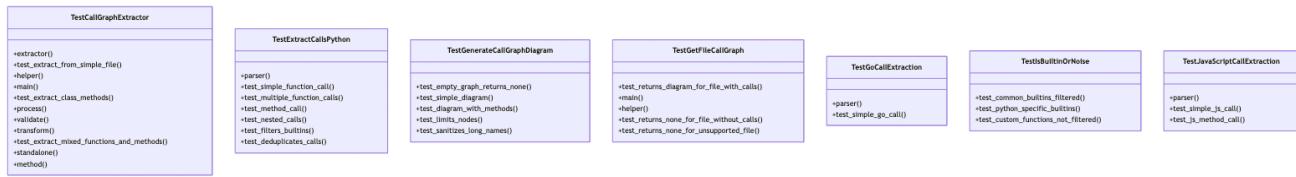
test_simple_go_call

```
def test_simple_go_call(parser)
```

Test extracting calls from Go function.

Parameter	Type	Default	Description
parser	-	-	-

Class Diagram



Call Graph



Relevant Source Files

- `tests/test_callgraph.py:20–39`

See Also

- [test_api_docs](#) - shares 5 dependencies
- [test_parser](#) - shares 4 dependencies

File Overview

This file contains tests for the `CodeChunker` class, which is responsible for breaking down code files into logical chunks such as functions, classes, and modules. The tests cover various scenarios including chunking Python and JavaScript files, extracting names and docstrings, setting line numbers, and ensuring unique IDs for chunks.

Classes

TestCodeChunker

The `TestCodeChunker` class contains a suite of tests to verify the functionality of the `CodeChunker` class. It includes methods to test different aspects of code chunking, such as handling different file types, extracting metadata like function and class names, and ensuring correct line number assignments.

Methods

- **`setup_method`**: Initializes a `CodeChunker` instance for each test.
- **`test_chunk_python_file`**: Tests chunking a Python file and verifies that the resulting chunks are correctly formed.
- **`test_chunk_extraction_function_names`**: Tests that function names are correctly extracted from a Python file.
- **`test_chunk_extraction_class_names`**: Tests that class names are correctly extracted from a Python file.
- **`test_chunk_extraction_docstrings`**: Tests that docstrings are correctly extracted from functions.
- **`test_chunk_javascript_file`**: Tests chunking a JavaScript file and verifies the resulting chunks.
- **`test_chunk_sets_line_numbers`**: Tests that line numbers are correctly assigned to chunks.
- **`test_chunk_generates_unique_ids`**: Tests that each chunk is assigned a unique ID.
- **`test_chunk_unsupported_file_returns_empty`**: Tests that unsupported file types return no chunks.

Functions

setup_method

Initializes a `CodeChunker` instance for each test method in the class.

Parameters: None

Returns: None

test_chunk_python_file

Tests chunking a Python file and verifies that the resulting chunks are correctly formed.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

test_chunk_extraction_function_names

Tests that function names are correctly extracted from a Python file.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

test_chunk_extraction_class_names

Tests that class names are correctly extracted from a Python file.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

test_chunk_extraction_docstrings

Tests that docstrings are correctly extracted from functions in a Python file.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

test_chunk_javascript_file

Tests chunking a JavaScript file and verifies the resulting chunks.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

test_chunk_sets_line_numbers

Tests that line numbers are correctly assigned to chunks in a Python file.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

test_chunk_generates_unique_ids

Tests that each chunk is assigned a unique ID.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

test_chunk_unsupported_file_returns_empty

Tests that unsupported file types return no chunks.

Parameters: - `tmp_path` : A temporary directory path provided by pytest for test file creation.

Returns: None

Usage Examples

To run the tests in this file, use pytest:

```
pytest tests/test_chunker.py
```

Each test method creates a temporary file with specific code content and then calls the `chunk_file` method of the `CodeChunker` instance to generate chunks. The tests assert various properties of the resulting chunks, such as their type, name, docstring, line numbers, and uniqueness of IDs.

Related Components

This file works with the following components:

- **CodeChunker**: The `main` class being tested, which performs the actual chunking of code files.
- **ChunkType**: An enum that defines the types of chunks that can be generated (e.g., FUNCTION, CLASS).
- **Language**: An enum that specifies the programming language of the code being chunked.

API Reference

class TestCodeChunker

Test suite for `CodeChunker`.

Methods:

`setup_method`

```
def setup_method()
```

Set up test fixtures.

test_chunk_python_file

```
def test_chunk_python_file(tmp_path)
```

Test chunking a Python file.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_chunk_extraction_function_names

```
def test_chunk_extraction_function_names(tmp_path)
```

Test that function names are extracted.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_chunk_extraction_class_names

```
def test_chunk_extraction_class_names(tmp_path)
```

Test that class names are extracted.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_chunk_extraction_docstrings

```
def test_chunk_extraction_docstrings(tmp_path)
```

Test that docstrings are extracted.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_chunk_javascript_file

```
def test_chunk_javascript_file(tmp_path)
```

Test chunking a JavaScript file.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_chunk_sets_line_numbers

```
def test_chunk_sets_line_numbers(tmp_path)
```

Test that line numbers are set correctly.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_chunk_generates_unique_ids

```
def test_chunk_generates_unique_ids(tmp_path)
```

Test that chunk IDs are unique.

Parameter	Type	Default	Description
tmp_path	-	-	-

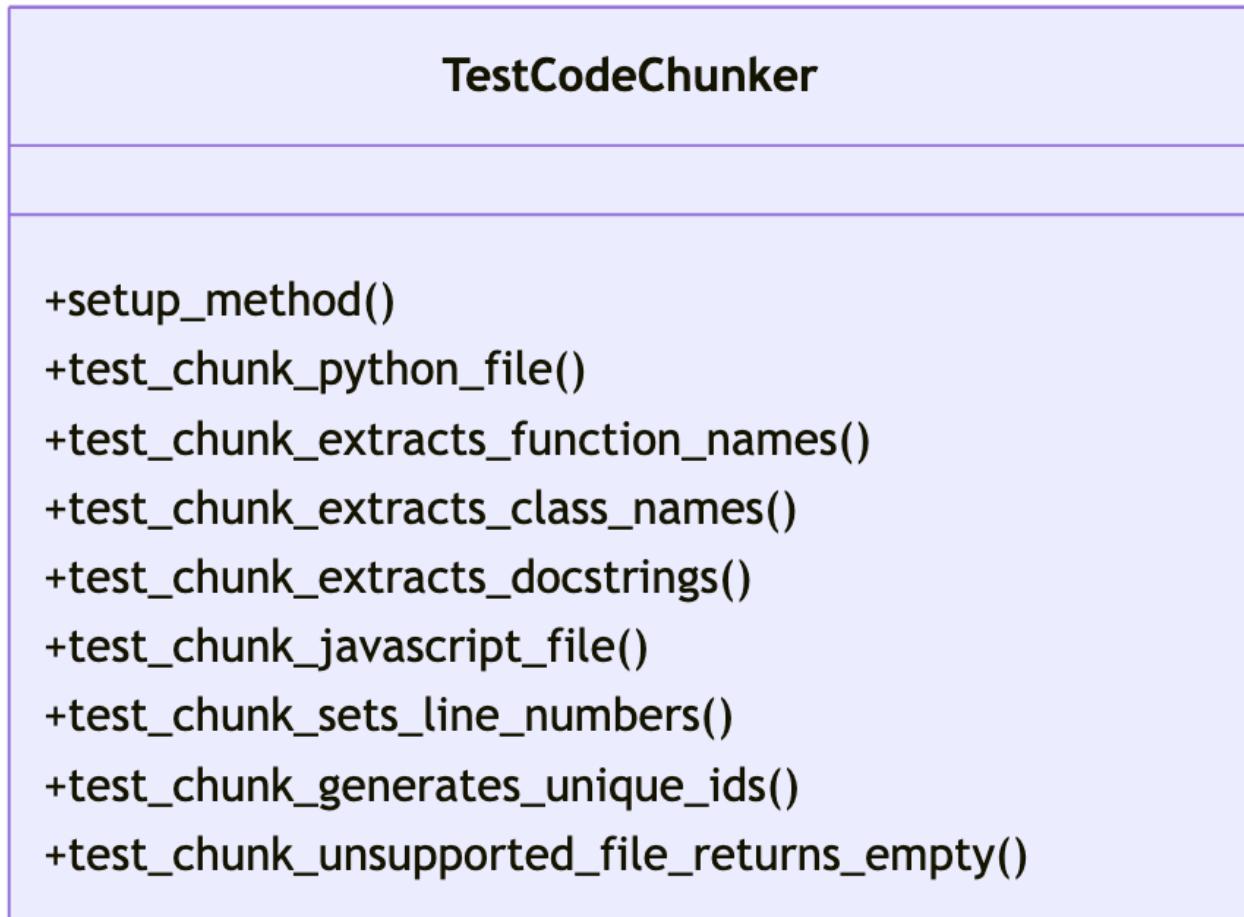
test_chunk_unsupported_file_returns_empty

```
def test_chunk_unsupported_file_returns_empty(tmp_path)
```

Test that unsupported files return no chunks.

Parameter	Type	Default	Description
tmp_path	-	-	-

Class Diagram



Call Graph



Relevant Source Files

- `tests/test_chunker.py`

See Also

- [models](#) - dependency
- [chunker](#) - dependency
- [api_docs](#) - shares 3 dependencies
- [test_api_docs](#) - shares 3 dependencies

Test Configuration File Documentation

File Overview

This file contains the test suite for the configuration system of the local_deepwiki application. It verifies that the `Config` class and related functions behave correctly with default values and specific configurations.

Classes

`TestConfig`

Purpose: Test suite for validating the `Config` class functionality and default configuration values.

Key Methods: - `test_default_config()` : Validates default configuration values -
`test_embedding_config()` : Tests embedding-specific configuration settings

Functions

`get_config()`

Parameters: None **Return Value:** Current configuration instance **Purpose:** Retrieves the current configuration instance

`set_config()`

Parameters: `config` (Config instance) **Return Value:** None **Purpose:** Sets the global configuration instance

Usage Examples

Basic Configuration Testing

```
# Test default configuration
def test_default_config(self):
    config = Config()
    assert config.embedding.provider == "local"
    assert config.llm.provider == "ollama"
    assert "python" in config.parsing.languages
    assert config.chunking.max_chunk_tokens == 512
```

Embedding Configuration Testing

```
# Test embedding-specific settings
def test_embedding_config(self):
    config = Config()
    assert config.embedding.local.model == "all-MiniLM-L6-v2"
    assert config.embedding.openai.model == "text-embedding-3-small"
```

Dependencies

This file imports: - `pathlib.Path` - For path manipulation - `pytest` - Testing framework - `local_deepwiki.config.Config` - Main configuration class - `local_deepwiki.config.get_config` - Function to retrieve config - `local_deepwiki.config.set_config` - Function to set config

The tests validate the configuration system's default values and ensure proper initialization of various configuration components including embedding providers, LLM providers, parsing languages, and chunking settings.

Class Diagram

TestConfig

```
+test_default_config()  
+test_embedding_config()  
+test_llm_config()  
+test_parsing_config()  
+test_get_wiki_path()  
+test_get_vector_db_path()  
+test_global_config()  
+test_set_config()
```

See Also

- [config](#) - dependency
- [test_chunker](#) - shares 2 dependencies
- [chunker](#) - shares 2 dependencies

- [**test_parser**](#) - shares 2 dependencies

File Overview

This file contains unit tests for various diagram generation functions within the `local_deepwiki.generators.diagrams` module. The tests cover functions that generate class diagrams, dependency graphs, module overviews, language pie charts, and sequence diagrams from code chunks and import information.

Classes

TestSanitizeMermaidName

Tests for the `_sanitize_mermaid_name` function, which ensures names used in Mermaid diagrams are valid.

TestExtractClassAttributes

Tests for the `_extract_class_attributes` function, which extracts attributes from class definitions.

TestExtractMethodSignature

Tests for the `_extract_method_signature` function, which extracts method signatures from code.

TestGenerateClassDiagram

Tests for the `generate_class_diagram` function, which generates Mermaid class diagrams from code chunks.

Methods

- `test_generates_diagram_with_class` : Tests diagram generation with a single class.

- `test_generates_diagram_with_multiple_classes` : Tests diagram generation with multiple classes.
- `test_handles_inheritance` : Tests diagram generation with class inheritance.
- `test_handles_method_signatures` : Tests diagram generation with method signatures.
- `test_handles_attributes` : Tests diagram generation with class attributes.

TestGenerateDependencyGraph

Tests for the `generate_dependency_graph` function, which generates Mermaid flowcharts showing module dependencies.

Methods

- `test_generates_flowchart` : Tests basic flowchart generation from code chunks.
- `test_handles_multiple_imports` : Tests handling of multiple import statements.

TestGenerateModuleOverview

Tests for the `generate_module_overview` function, which generates Mermaid diagrams showing the structure of a module.

Methods

- `test_generates_diagram` : Tests module overview generation from index status data.

TestPathToModule

Tests for the `_path_to_module` function, which converts file paths to module names.

Methods

- `test Converts_simple_path` : Tests basic path conversion.
- `test_skips_init_files` : Tests that `__init__.py` files return `None`.

- `test_skips_non_python` : Tests that non-Python files return `None`.

TestGenerateSequenceDiagram

Tests for the `generate_sequence_diagram` function, which generates Mermaid sequence diagrams from call graphs.

Methods

- `test_generates_sequence` : Tests sequence diagram generation from a call graph.
- `test_returns_none_for_empty_call_graph` : Tests that an empty call graph returns `None`.

Functions

sanitize_mermaid_name

Sanitizes names for use in Mermaid diagrams by replacing invalid characters.

Parameters

- `name` (`str`): The name to sanitize.

Returns

- `str` : The sanitized name.

generate_class_diagram

Generates a Mermaid class diagram from a list of code chunks.

Parameters

- `chunks` (`list[CodeChunk]`): The code chunks to analyze.

Returns

- `str` : The Mermaid diagram as a string.

generate_dependency_graph

Generates a Mermaid flowchart showing dependencies between modules.

Parameters

- `chunks` (list[CodeChunk]): The code chunks containing import statements.

Returns

- `str` : The Mermaid flowchart as a string.

generate_module_overview

Generates a Mermaid diagram showing the structure of a module based on index status.

Parameters

- `status` (IndexStatus): The index status information.

Returns

- `str` : The Mermaid diagram as a string.

generate_language_pie_chart

Generates a Mermaid pie chart showing the distribution of languages in a repository.

Parameters

- `languages` (dict[str, int]): A dictionary mapping language names to counts.

Returns

- `str` : The Mermaid pie chart as a string.

generate_sequence_diagram

Generates a Mermaid sequence diagram from a call graph.

Parameters

- `call_graph` (`dict[str, list[str]]`): A dictionary mapping function names to their called functions.
- `main_function` (`str`): The `main` function to start the diagram from.

Returns

- `str` : The Mermaid sequence diagram as a string or `None` if the call graph is empty.

_extract_class_attributes

Extracts attributes from a class definition.

Parameters

- `class_def` (`str`): The class definition string.

Returns

- `list[str]` : A list of attribute names.

_extract_method_signature

Extracts a method signature from a method definition.

Parameters

- `method_def` (str): The method definition string.

Returns

- `str` : The method signature.

`_find_circular_dependencies`

Finds circular dependencies in a list of import statements.

Parameters

- `imports` (list[str]): A list of import statements.

Returns

- `list[tuple[str, str]]` : A list of tuples representing circular dependencies.

`_path_to_module`

Converts a file path to a module name.

Parameters

- `path` (str): The file path.

Returns

- `str | None` : The module name or `None` if the path is not valid.

`_parse_import_line`

Parses an import line to extract the module name.

Parameters

- `line` (str): The import line.

Returns

- `str | None` : The module name or `None` if parsing fails.

Usage Examples

Generating a Class Diagram

```
chunks = [
    CodeChunk(
        id="1",
        file_path="test.py",
        content="class MyClass:\n            def method(self): pass",
        chunk_type=ChunkType.CLASS,
        language=Language.PYTHON,
        start_line=1,
        end_line=2,
        name="MyClass",
        metadata={}
    )
]
diagram = generate_class_diagram(chunks)
```

Generating a Dependency Graph

```
chunks = [
    CodeChunk(
        id="1",
        file_path="src/local_deepwiki/core/parser.py",
        content="from local_deepwiki.models import ChunkType",
        chunk_type=ChunkType.IMPORT,
        language=Language.PYTHON,
        start_line=1,
        end_line=1,
    )
]
graph = generate_dependency_graph(chunks)
```

Generating a Module Overview

```
status = IndexStatus(
    repo_path="/test",
    indexed_at=1234567890.0,
    total_files=3,
    total_chunks=10,
    languages={"python": 3},
    files=[
        FileInfo(path="src/core/parser.py", language="python", hash="a", chunk_count=5, size_bytes=1),
        FileInfo(path="src/core/chunker.py", language="python", hash="b", chunk_count=3, size_bytes=1)
    ]
)
overview = generate_module_overview(status)
```

Generating a Sequence Diagram

```
call_graph = {
    "main": ["process", "validate"],
    "process": ["transform"],
    "validate": [],
    "transform": [],
}
diagram = generate_sequence_diagram(call_graph, "main")
```

Related Components

This file works with the following components:

- `local_deepwiki.generators.diagrams` module: Contains the actual diagram generation functions.
- `local_deepwiki.models` : Provides data models like `ChunkType`, `CodeChunk`, `Language`, `IndexStatus`, and `FileInfo` used in tests.

API Reference

`class TestSanitizeMermaidName`

Tests for [sanitize_mermaid_name](#) function.

Methods:

`test_basic_name`

```
def test_basic_name()
```

Test basic name passes through.

`test_replaces_brackets`

```
def test_replaces_brackets()
```

Test angle brackets are replaced.

`test_replaces_square_brackets`

```
def test_replaces_square_brackets()
```

Test square brackets are replaced.

test_replaces_dots

```
def test_replaces_dots()
```

Test dots are replaced.

test_replaces_hyphens

```
def test_replaces_hyphens()
```

Test hyphens are replaced.

test_replaces_colons

```
def test_replaces_colons()
```

Test colons are replaced.

test_prefixes_digit

```
def test_prefixes_digit()
```

Test names starting with digits get prefixed.

class TestExtractClassAttributes

Tests for _extract_class_attributes function.

Methods:**test_extraction_type_annotations**

```
def test_extraction_type_annotations()
```

Test extraction of class-level type annotations.

test_extraction_init_assignments

```
def test_extraction_init_assignments()
```

Test extraction from `init` assignments.

test_marks_private_attributes

```
def test_marks_private_attributes()
```

Test private attributes get - prefix.

class TestExtractMethodSignature

Tests for `_extract_method_signature` function.

Methods:**test_extraction_return_type**

```
def test_extraction_return_type()
```

Test extraction of return type.

test_extraction_parameters

```
def test_extraction_parameters()
```

Test extraction of parameters.

test_excludes_self

```
def test_excludes_self()
```

Test self parameter is excluded.

test_limits_parameters

```
def test_limits_parameters()
```

Test long parameter lists are truncated.

test_returns_none_for_invalid

```
def test_returns_none_for_invalid()
```

Test returns None for non-def content.

class TestClassInfo

Tests for [ClassInfo](#) dataclass.

Methods:**test_basic_class_info**

```
def test_basic_class_info()
```

Test basic [ClassInfo](#) creation.

test_abstract_class

```
def test_abstract_class()
```

Test abstract class flag.

class TestGenerateClassDiagram

Tests for [generate_class_diagram](#) function.

Methods:**test_generates_diagram_with_class**

```
def test_generates_diagram_with_class()
```

Test diagram generation with a single class.

test_returns_none_for_empty_classes

```
def test_returns_none_for_empty_classes()
```

Test returns None when classes have no content.

test_shows_inheritance

```
def test_shows_inheritance()
```

Test inheritance relationships are shown.

test_marks_dataclass

```
def test_marks_dataclass()
```

Test dataclass annotation is shown.

test_shows_method_visibility

```
def test_shows_method_visibility()
```

Test private methods are marked with -.

class TestGenerateDependencyGraph

Tests for [generate_dependency_graph](#) function.

Methods:**test_generates_flowchart**

```
def test_generates_flowchart()
```

Test basic flowchart generation.

test_returns_none_for_no_imports

```
def test_returns_none_for_no_imports()
```

Test returns None when no imports.

class TestFindCircularDependencies

Tests for _find_circular_dependencies function.

Methods:**test_finds_direct_cycle**

```
def test_finds_direct_cycle()
```

Test detection of A -> B -> A cycle.

test_finds_longer_cycle

```
def test_finds_longer_cycle()
```

Test detection of A -> B -> C -> A cycle.

test_no_cycle

```
def test_no_cycle()
```

Test no false positives for acyclic graph.

class TestPathToModule

Tests for _path_to_module function.

Methods:**test Converts simple path**

```
def test Converts_simple_path()
```

Test basic path conversion.

test_skips_init_files

```
def test_skips_init_files()
```

Test `init.py` files return None.

```
test_skips_non_python
```

```
def test_skips_non_python()
```

Test non-Python files return None.

```
class TestParseImportLine
```

Tests for `_parse_import_line` function.

Methods:

```
test_parses_from_import
```

```
def test_parses_from_import()
```

Test from X import Y parsing.

```
test_ignores_external
```

```
def test_ignores_external()
```

Test external imports return None.

```
test_parses_import_statement
```

```
def test_parses_import_statement()
```

Test import X parsing.

```
class TestGenerateModuleOverview
```

Tests for [generate_module_overview](#) function.

Methods:

test_generates_diagram

```
def test_generates_diagram()
```

Test module overview generation.

test_returns_none_for_empty

```
def test_returns_none_for_empty()
```

Test returns None when no files.

class TestGenerateLanguagePieChart

Tests for [generate_language_pie_chart](#) function.

Methods:**test_generates_pie_chart**

```
def test_generates_pie_chart()
```

Test pie chart generation.

test_returns_none_for_no_languages

```
def test_returns_none_for_no_languages()
```

Test returns None when no languages.

class TestGenerateSequenceDiagram

Tests for [generate_sequence_diagram](#) function.

Methods:**test_generates_sequence**

```
def test_generates_sequence()
```

Test sequence diagram generation.

test_returns_none_for_empty

```
def test_returns_none_for_empty()
```

Test returns None for empty call graph.

test_auto_selects_entry_point

```
def test_auto_selects_entry_point()
```

Test auto-selects entry point when not specified.

Class Diagram



Call Graph

Relevant Source Files

- tests/test_diagrams.py:22–51

See Also

- [diagrams](#) - dependency
 - [test_api_docs](#) - shares 2 dependencies

File Overview

This file contains tests for the HTML export functionality of the deepwiki tool. It verifies that wiki content is correctly converted to HTML, including proper directory structure, content rendering, navigation elements, and theme support.

Classes

TestRenderMarkdown

Tests for markdown rendering functionality.

Methods

- `test_basic_markdown()` : Tests basic markdown conversion to HTML
- `test_code_blocks()` : Tests fenced code block rendering
- `test_tables()` : Tests table rendering

TestExtractTitle

Tests for title extraction functionality.

Methods

- `test_h1_title(tmp_path: Path)` : Tests extracting title from H1 markdown header
- `test_bold_title(tmp_path: Path)` : Tests extracting title from bold markdown text
- `test_fallback_to_filename(tmp_path: Path)` : Tests fallback to filename when no title found

TestHtmlExporter

Tests for the [HtmlExporter](#) class functionality.

Methods

- `sample_wiki(tmp_path: Path) -> Path` : Creates a sample wiki structure for testing
- `test_exportCreatesOutputDirectory(sample_wiki: Path, tmp_path: Path)` : Tests that export creates the output directory
- `test_exportCreatesHtmlFiles(sample_wiki: Path, tmp_path: Path)` : Tests that export creates HTML files for each markdown file
- `test_exportCopiesSearchJson(sample_wiki: Path, tmp_path: Path)` : Tests that export copies search.json
- `test_htmlContainsContent(sample_wiki: Path, tmp_path: Path)` : Tests that HTML files contain the converted content
- `test_htmlContainsToc(sample_wiki: Path, tmp_path: Path)` : Tests that HTML files contain the table of contents
- `test_htmlHasRelativeLinks(sample_wiki: Path, tmp_path: Path)` : Tests that HTML files use relative links
- `test_htmlHasBreadcrumbForNestedPages(sample_wiki: Path, tmp_path: Path)` : Tests that nested pages have breadcrumb navigation
- `test_htmlHasThemeToggle(sample_wiki: Path, tmp_path: Path)` : Tests that HTML files have theme toggle functionality

TestExportToHtml

Tests for the [export_to_html](#) convenience function.

Methods

- `test_defaultOutputPath(simple_wiki: Path)` : Tests that default output path is {wiki_path}_html

Functions

export_to_html

Convenience function for exporting wiki to HTML.

Parameters

- `wiki_path: Path` : Path to the wiki directory
- `output_path: Path | None` : Optional output directory path

Returns

- `int` : Number of files exported

Usage Examples

Testing HTML Export

```
# Test that export creates the output directory
def test_exportCreatesOutputDirectory(self, sample_wiki: Path, tmp_path: Path):
    output_path = tmp_path / "html_output"
    exporter = HtmlExporter(sample_wiki, output_path)
    exporter.export()

    assert output_path.exists()
    assert output_path.is_dir()
```

Testing HTML Content

```
# Test that HTML files contain the converted content
def test_html_contains_content(self, sample_wiki: Path, tmp_path: Path):
    output_path = tmp_path / "html_output"
    exporter = HtmlExporter(sample_wiki, output_path)
    exporter.export()

    html = (output_path / "index.html").read_text()
    assert "Overview" in html
    assert "Welcome to the wiki" in html
    assert "<!DOCTYPE html>" in html
```

Testing Markdown Rendering

```
# Test basic markdown conversion
def test_basic_markdown(self):
    md = "# Hello\n\nThis is a paragraph."
    html = render_markdown(md)
    assert "<h1>" in html
    assert "Hello" in html
    assert "<p>" in html
```

Related Components

This file works with the following components from `local_deepwiki.export.html`:

- `HtmlExporter` : Main class for exporting wiki to HTML
- `export_to_html` : Convenience function for exporting wiki to HTML
- `render_markdown` : Function for converting markdown to HTML
- `extract_title` : Function for extracting titles from markdown files

API Reference

class `TestRenderMarkdown`

Tests for markdown rendering.

Methods:

`test_basic_markdown`

```
def test_basic_markdown()
```

Test basic markdown conversion.

`test_code_blocks`

```
def test_code_blocks()
```

Test fenced code blocks.

`test_tables`

```
def test_tables()
```

Test markdown tables.

`test_mermaid_blocks`

```
def test_mermaid_blocks()
```

Test mermaid code blocks are preserved.

class `TestExtractTitle`

Tests for title extraction.

Methods:

test_h1_title

```
def test_h1_title(tmp_path: Path)
```

Test extracting H1 title.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_bold_title

```
def test_bold_title(tmp_path: Path)
```

Test extracting bold title.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_fallback_to_filename

```
def test_fallback_to_filename(tmp_path: Path)
```

Test fallback to filename when no title found.

Parameter	Type	Default	Description
tmp_path	Path	-	-

class TestHtmlExporter

Tests for [HtmlExporter](#) class.

Methods:

sample_wiki

```
def sample_wiki(tmp_path: Path) -> Path
```

Create a sample wiki structure for testing.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_exportCreatesOutputDirectory

```
def test_exportCreatesOutputDirectory(sample_wiki: Path, tmp_path: Path)
```

Test that export creates the output directory.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_exportCreatesHtmlFiles

```
def test_exportCreatesHtmlFiles(sample_wiki: Path, tmp_path: Path)
```

Test that export creates HTML files for each markdown file.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_exportCopiesSearchJson

```
def test_exportCopiesSearchJson(sample_wiki: Path, tmp_path: Path)
```

Test that export copies search.json.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

`test_html_contains_content`

```
def test_html_contains_content(sample_wiki: Path, tmp_path: Path)
```

Test that HTML files contain the converted content.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

`test_html_contains_toc`

```
def test_html_contains_toc(sample_wiki: Path, tmp_path: Path)
```

Test that HTML files contain the TOC.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

`test_html_has_relative_links`

```
def test_html_has_relative_links(sample_wiki: Path, tmp_path: Path)
```

Test that HTML files use relative links.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_html_has_breadcrumb_for_nested_pages

```
def test_html_hasBreadcrumbForNestedPages(sample_wiki: Path, tmp_path: Path)
```

Test that nested pages have breadcrumb navigation.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

test_html_has_theme_toggle

```
def test_html_hasThemeToggle(sample_wiki: Path, tmp_path: Path)
```

Test that HTML files have theme toggle functionality.

Parameter	Type	Default	Description
sample_wiki	Path	-	-
tmp_path	Path	-	-

class TestExportToHtml

Tests for the [export_to_html](#) convenience function.

Methods:

simple_wiki

```
def simple_wiki(tmp_path: Path) -> Path
```

Create a simple wiki for testing.

Parameter	Type	Default	Description
tmp_path	Path	-	-

test_default_output_path

```
def test_default_output_path(simple_wiki: Path)
```

Test that default output path is {wiki_path}_html.

Parameter	Type	Default	Description
simple_wiki	Path	-	-

test_custom_output_path

```
def test_custom_output_path(simple_wiki: Path, tmp_path: Path)
```

Test that custom output path is used.

Parameter	Type	Default	Description
simple_wiki	Path	-	-
tmp_path	Path	-	-

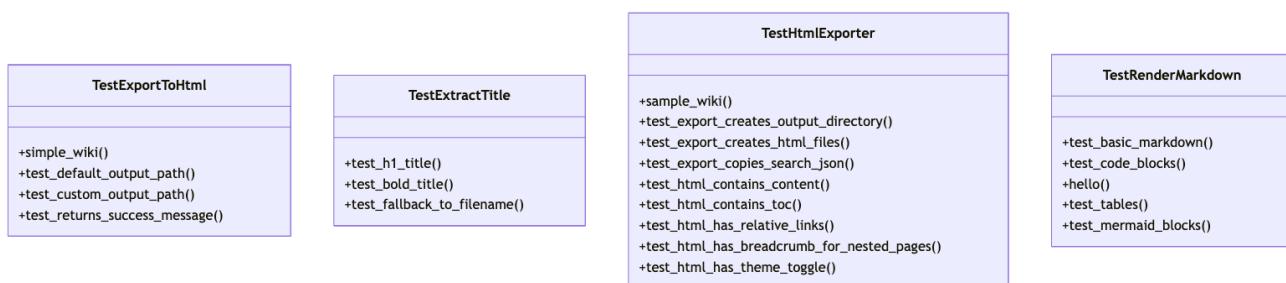
test_returns_success_message

```
def test_returns_success_message(simple_wiki: Path, tmp_path: Path)
```

Test that export returns a success message.

Parameter	Type	Default	Description
simple_wiki	Path	-	-
tmp_path	Path	-	-

Class Diagram



Call Graph



Relevant Source Files

- tests/test_html_export.py:16-47

See Also

- [html](#) - dependency
 - [test_incremental_wiki](#) - shares 3 dependencies

File Overview

This file contains tests for the incremental wiki generation functionality. It tests the models and helper methods used to track wiki page generation status and determine whether regeneration is needed based on source file changes.

Classes

TestWikiPageStatus

Test class for the WikiPageStatus model.

Methods

test_create_page_status

Test creating a WikiPageStatus.

TestWikiGenerationStatus

Test class for the WikiGenerationStatus model.

Methods

test_create_generation_status

Test creating a WikiGenerationStatus.

test_generation_status_with_pages

Test generation status with page statuses.

TestWikiGeneratorHelpers

Test class for helper methods in [WikiGenerator](#).

Methods

mock_wiki_generator

Create a mock [WikiGenerator](#).

test_compute_content_hash

Test computing content hash.

test_needs_regeneration_no_previous_status

Test needs_regeneration when no previous status exists.

test_needs_regeneration_page_not_in_status

Test needs_regeneration when page not in previous status.

test_needs_regeneration_source_hash_changed

Test needs_regeneration when source file hash changed.

test_needs_regeneration_no_changes

Test needs_regeneration when nothing changed.

test_needs_regeneration_source_files_changed

Test needs_regeneration when source files list changed.

test_record_page_status

Test recording page status.

TestWikiStatusPersistence

Test class for wiki status file persistence.

Methods

`test_save_and_load_wiki_status`

Test saving and loading wiki status.

TestLoadExistingPage

Test class for loading existing wiki pages.

Methods

`test_load_existing_page`

Test loading an existing page from disk.

Functions

mock_wiki_generator

Create a mock [WikiGenerator](#).

Returns: A mock [WikiGenerator](#) instance with predefined attributes.

Usage Examples

Testing WikiPageStatus Creation

```
def test_create_page_status(self):
    status = WikiPageStatus(
        path="files/test.md",
        source_files=["src/test.py"],
        source_hashes={"src/test.py": "abc123"},
        content_hash="def456",
        generated_at=time.time(),
    )
    assert status.path == "files/test.md"
```

Testing WikiGenerationStatus Creation

```
def test_create_generation_status(self):
    status = WikiGenerationStatus(
        repo_path="/path/to/repo",
        generated_at=time.time(),
        total_pages=5,
        index_status_hash="abc123",
        pages={},
    )
    assert status.repo_path == "/path/to/repo"
    assert status.total_pages == 5
```

Testing Page Status Recording

```
def test_record_page_status(self, mock_wiki_generator):
    page = WikiPage(
        path="test.md",
        title="Test",
        content="# Test\nContent here",
        generated_at=time.time(),
    )
    mock_wiki_generator._record_page_status(page, ["src/test.py"])
    assert "test.md" in mock_wiki_generator._page_statuses
```

Related Components

This file works with the following components:

- [WikiGenerator](#) class
- WikiPageStatus model
- WikiGenerationStatus model
- WikiPage model
- FileInfo model
- IndexStatus model
- Language model

The tests use mocking to isolate the functionality being tested, particularly the [WikiGenerator](#) class and its helper methods.

API Reference

class `TestWikiPageStatus`

Test WikiPageStatus model.

Methods:

`test_create_page_status`

```
def test_create_page_status()
```

Test creating a WikiPageStatus.

`test_page_status_multiple_sources`

```
def test_page_status_multiple_sources()
```

Test page status with multiple source files.

class `TestWikiGenerationStatus`

Test WikiGenerationStatus model.

Methods:

`test_create_generation_status`

```
def test_create_generation_status()
```

Test creating a WikiGenerationStatus.

`test_generation_status_with_pages`

```
def test_generation_status_with_pages()
```

Test generation status with page statuses.

class `TestWikiGeneratorHelpers`

Test [WikiGenerator](#) helper methods.

Methods:

`mock_wiki_generator`

```
def mock_wiki_generator()
```

Create a mock [WikiGenerator](#).

`test_compute_content_hash`

```
def test_compute_content_hash(mock_wiki_generator)
```

Test content hash computation.

Parameter	Type	Default	Description
mock_wiki_generator	-	-	-

`test_needs_regeneration_no_previous_status`

```
def test_needs_regeneration_no_previous_status(mock_wiki_generator)
```

Test needs_regeneration when no previous status exists.

Parameter	Type	Default	Description
mock_wiki_generator	-	-	-

`test_needs_regeneration_page_not_in_status`

```
def test_needs_regeneration_page_not_in_status(mock_wiki_generator)
```

Test needs_regeneration when page not in previous status.

Parameter	Type	Default	Description
mock_wiki_generator	-	-	-

`test_needs_regeneration_source_hash_changed`

```
def test_needs_regeneration_source_hash_changed(mock_wiki_generator)
```

Test needs_regeneration when source file hash changed.

Parameter	Type	Default	Description
mock_wiki_generator	-	-	-

test_needs_regeneration_no_changes

```
def test_needs_regeneration_no_changes(mock_wiki_generator)
```

Test needs_regeneration when nothing changed.

Parameter	Type	Default	Description
mock_wiki_generator	-	-	-

test_needs_regeneration_source_files_changed

```
def test_needs_regeneration_source_files_changed(mock_wiki_generator)
```

Test needs_regeneration when source files list changed.

Parameter	Type	Default	Description
mock_wiki_generator	-	-	-

test_record_page_status

```
def test_record_page_status(mock_wiki_generator)
```

Test recording page status.

Parameter	Type	Default	Description
mock_wiki_generator	-	-	-

class TestWikiStatusPersistence

Test wiki status file persistence.

Methods:**test_save_and_load_wiki_status**

```
async def test_save_and_load_wiki_status(tmp_path)
```

Test saving and loading wiki status.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_load_missing_status

```
async def test_load_missing_status(tmp_path)
```

Test loading when status file doesn't exist.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_load_corrupted_status

```
async def test_load_corrupted_status(tmp_path)
```

Test loading when status file is corrupted.

Parameter	Type	Default	Description
tmp_path	-	-	-

class TestLoadExistingPage

Test loading existing wiki pages.

Methods:

test_load_existing_page

```
async def test_load_existing_page(tmp_path)
```

Test loading an existing page from disk.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_load_missing_page

```
async def test_load_missing_page(tmp_path)
```

Test loading a page that doesn't exist.

Parameter	Type	Default	Description
tmp_path	-	-	-

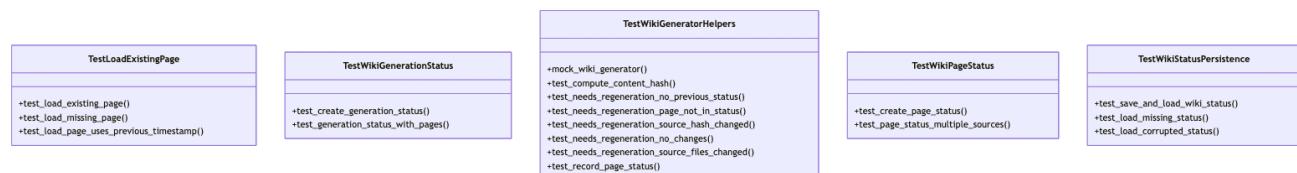
test_load_page_uses_previous_timestamp

```
async def test_load_page_uses_previous_timestamp(tmp_path)
```

Test that loaded page uses timestamp from previous status.

Parameter	Type	Default	Description
tmp_path	-	-	-

Class Diagram



Call Graph



Relevant Source Files

- tests/test_incremental_wiki.py:20-47

See Also

- [test_indexer](#) - shares 5 dependencies
 - [server](#) - shares 4 dependencies

File: test_search.py

File Overview

This file contains unit tests for the search-related functionality in the local_deepwiki project. It tests functions that extract information from wiki pages for search indexing, generate search entries, and write search indexes to disk. The tests ensure that search functionality works correctly and that the search index is properly structured.

The file is part of the test suite and works with the core [WikiPage](#) model and search generation components. It also tests the Flask web application's search endpoint.

Classes

TestWriteSearchIndex

Tests for the write_search_index function.

Methods: - `test_writes_json_file` : Tests that the search index is written to disk as a JSON file.

TestGenerateSearchEntry

Tests for the generate_search_entry function.

Methods: - `test_generates_complete_entry` : Tests that all fields (path, title, headings, terms, snippet) are properly populated in the search entry.

TestSearchJsonEndpoint

Tests for the Flask /search.json endpoint.

Methods: - `test_returns_search_index` : Tests that the /search.json endpoint returns the correct search index data.

TestGenerateSearchIndex

Tests for the generate_search_index function.

Methods: - `test_generates_index_for_multiple_pages` : Tests index generation with multiple wiki pages.

TestExtractHeadings

Tests for the extract_headings function.

Methods: - (Not fully shown in provided code, but would test heading extraction from wiki content)

TestExtractCodeTerms

Tests for the extract_code_terms function.

Methods: - (Not fully shown in provided code, but would test code term extraction from wiki content)

TestExtractSnippet

Tests for the extract_snippet function.

Methods: - (Not fully shown in provided code, but would test snippet extraction from wiki content)

Functions

extract_code_terms

Extracts code terms (identifiers, class names, etc.) from wiki page content.

Parameters: - `content` (str): The content of the wiki page

Returns: - `list[str]` : A list of code terms found in the content

extract_headings

Extracts headings from wiki page content.

Parameters: - `content` (str): The content of the wiki page

Returns: - `list[str]` : A list of heading strings found in the content

extract_snippet

Extracts a snippet from wiki page content.

Parameters: - `content` (str): The content of the wiki page

Returns: - `str` : A snippet of the content, typically the first paragraph or line

generate_search_entry

Generates a search entry for a single wiki page.

Parameters: - `page` ([WikiPage](#)): The wiki page to generate a search entry for

Returns: - `dict` : A dictionary containing search entry fields (path, title, headings, terms, snippet)

generate_search_index

Generates a complete search index for multiple wiki pages.

Parameters: - `pages` (`list[WikiPage]`): A list of wiki pages to index

Returns: - `list[dict]` : A list of search entries, one for each page

write_search_index

Writes the search index to a JSON file on disk.

Parameters: - `wiki_path` (Path): The path to the wiki directory - `pages` (`list[WikiPage]`): The list of wiki pages to index

Returns: - `Path` : The path to the written search index file

Usage Examples

Generate a search entry for a wiki page

```
from local_deepwiki.models import WikiPage
from local_deepwiki.generators.search import generate_search_entry

page = WikiPage(
    path="index.md",
    title="Home",
    content="# Home Page\n\nThis is the home page content.",
    generated_at=0,
)
entry = generate_search_entry(page)
print(entry)
# Output:
#   "path": "index.md",
#   "title": "Home",
#   "headings": ["Home Page"],
#   "terms": [],
#   "snippet": "This is the home page content."
# }
```

Generate a search index for multiple pages

```
from local_deepwiki.models import WikiPage
from local_deepwiki.generators.search import generate_search_index

pages = [
    WikiPage(
        path="index.md",
        title="Home",
        content="# Home Page\n\nWelcome to the home page.",
        generated_at=0,
    ),
    WikiPage(
        path="about.md",
        title="About",
        content="# About Us\n\nLearn about our company.",
        generated_at=0,
    ),
]
index = generate_search_index(pages)
print(index)
```

Write search index to disk

```
from pathlib import Path
from local_deepwiki.models import WikiPage
from local_deepwiki.generators.search import write_search_index

with tempfile.TemporaryDirectory() as tmpdir:
    wiki_path = Path(tmpdir)
    pages = [
        WikiPage(
            path="index.md",
            title="Home",
            content="# Home Page\n\nWelcome to the home page.",
            generated_at=0,
        ),
    ]
    result_path = write_search_index(wiki_path, pages)
    print(f"Search index written to: {result_path}")
```

Related Components

This file works with the [WikiPage](#) model to process wiki content for search indexing. It integrates with the search generation components in the generators.search module. The TestSearchJsonEndpoint class specifically tests the Flask web application's search endpoint, which depends on the search index generation functionality. The tests use the [create_app](#) function to set up a Flask test client for endpoint testing.

API Reference

class `TestExtractHeadings`

Tests for extract_headings function.

Methods:

`test_extract_h1_headings`

```
def test_extract_h1_headings()
```

Test extraction of h1 headings.

`test_extract_multiple_heading_levels`

```
def test_extract_multiple_heading_levels()
```

Test extraction of h1, h2, h3 headings.

`test_removes_markdown_formatting`

```
def test_removes_markdown_formatting()
```

Test that markdown formatting is stripped from headings.

`test_empty_content`

```
def test_empty_content()
```

Test with empty content.

class TestExtractCodeTerms

Tests for extract_code_terms function.

Methods:

test_extract_simple_terms

```
def test_extract_simple_terms()
```

Test extraction of simple backticked terms.

test_extract_qualified_names

```
def test_extract_qualified_names()
```

Test extraction of qualified names.

test_skips_long_code_blocks

```
def test_skips_long_code_blocks()
```

Test that long inline code is skipped.

test_empty_content

```
def test_empty_content()
```

Test with empty content.

class TestExtractSnippet

Tests for extract_snippet function.

Methods:

testExtracts_plain_text

```
def testExtracts_plain_text()
```

Test basic snippet extraction.

testRemoves_code_blocks

```
def testRemoves_code_blocks()
```

Test that code blocks are removed.

testRemoves_headings

```
def testRemoves_headings()
```

Test that headings are removed.

testRemoves_links_keeps_text

```
def testRemoves_links_keeps_text()
```

Test that link syntax is removed but text is kept.

testTruncates_long_content

```
def testTruncates_long_content()
```

Test that long content is truncated.

testEmpty_content

```
def testEmpty_content()
```

Test with empty content.

class TestGenerateSearchEntry

Tests for generate_search_entry function.

Methods:

test_generates_complete_entry

```
def test_generates_complete_entry()
```

Test that all fields are populated.

class TestGenerateSearchIndex

Tests for generate_search_index function.

Methods:

test_generates_index_for_multiple_pages

```
def test_generates_index_for_multiple_pages()
```

Test index generation with multiple pages.

class TestWriteSearchIndex

Tests for write_search_index function.

Methods:

test_writes_json_file

```
def test_writes_json_file()
```

Test that search index is written to disk.

class TestSearchJsonEndpoint

Tests for the Flask /search.json endpoint.

Methods:

test_returns_search_index

```
def test_returns_search_index()
```

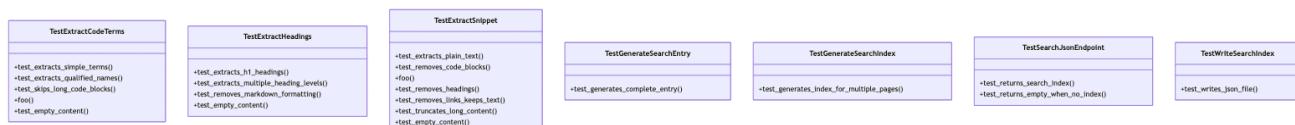
Test that /search.json returns the index.

test_returns_empty_when_no_index

```
def test_returns_empty_when_no_index()
```

Test that missing search.json returns empty array.

Class Diagram



Call Graph



See Also

- [models](#) - dependency
- [app](#) - dependency
- [test_incremental_wiki](#) - shares 4 dependencies

File Overview

This file, `tests/test_see_also.py`, contains unit tests for the see also section generation functionality within the `local_deepwiki` project. It tests various components related to analyzing file relationships and generating "See Also" sections for documentation pages.

Classes

TestRelationshipAnalyzer

Tests for the [RelationshipAnalyzer](#) class, which is responsible for analyzing file import relationships.

Methods

- `test_analyze_python_imports` : Tests the analysis of Python import statements.
- `test_get_relationships_imports` : Tests retrieving imported files from a given file.
- `test_get_relationships_imported_by` : Tests retrieving files that import a given file.
- `test_ignores_non_import_chunks` : Tests that non-import chunks are ignored during analysis.
- `test_shared_dependencies` : Tests shared dependencies between files.

TestBuildFileToWikiMap

Tests for the [build_file_to_wiki_map](#) function.

Methods

- `test_builds_map` : Tests building a mapping from file paths to wiki page paths.

TestGenerateSeeAlsoSection

Tests for the [generate_see_also_section](#) function.

Methods

- `test_generates_section_with_importers` : Tests generating a See Also section with files that import the current file.

TestRelativePath

Tests for the `_relative_path` function.

Methods

- `test_same_directory` : Tests relative path calculation in the same directory.
- `test_parent_directory` : Tests relative path calculation to a parent directory.
- `test_sibling_directory` : Tests relative path calculation to a sibling directory.

TestAddSeeAlsoSections

Tests for the `add_see_also_sections` function.

Methods

- `test_adds_sections_to_file_pages` : Tests that See Also sections are added to file documentation pages.

Functions

generate_see_also_section

Generates a "See Also" section for a given file based on its relationships.

Parameters

- `relationships` ([FileRelationships](#)): The file relationships object containing import and imported_by information.
- `file_to_wiki` (dict): A mapping of file paths to wiki page paths.
- `file_path` (str): The path of the file for which to generate the section.

Returns

- `str` : The generated "See Also" section content.

add_see_also_sections

Adds "See Also" sections to a list of wiki pages.

Parameters

- `analyzer` ([RelationshipAnalyzer](#)): The analyzer to use for determining relationships.
- `wiki_pages` (list): List of [WikiPage](#) objects to process.
- `file_to_wiki` (dict): A mapping of file paths to wiki page paths.

Returns

- `list` : The list of [WikiPage](#) objects with "See Also" sections added.

_relative_path

Calculates the relative path from one file to another.

Parameters

- `from_path` (str): The source file path.
- `to_path` (str): The target file path.

Returns

- `str` : The relative path from `from_path` to `to_path`.

Usage Examples

Using `generate_see_also_section`

```
from local_deepwiki.generators.see_also import generate_see_also_section, FileRelationships

relationships = FileRelationships(
    file_path="src/local_deepwiki/core/chunker.py",
    imported_by={"src/local_deepwiki/core/indexer.py"},
)
file_to_wiki = {
    "src/local_deepwiki/core/chunker.py": "files/src/local_deepwiki/core/chunker.md",
    "src/local_deepwiki/core/indexer.py": "files/src/local_deepwiki/core/indexer.md",
}
section = generate_see_also_section(relationships, file_to_wiki, "src/local_deepwiki/core/chunker.py")
```

Using `add_see_also_sections`

```
from local_deepwiki.generators.see_also import add_see_also_sections, RelationshipAnalyzer
from local_deepwiki.models import WikiPage

analyzer = RelationshipAnalyzer()
wiki_pages = [WikiPage(...)]
file_to_wiki = {...}
updated_pages = add_see_also_sections(analyzer, wiki_pages, file_to_wiki)
```

Related Components

This file works with the following components:

- [RelationshipAnalyzer](#) : Analyzes file import relationships.
- [FileRelationships](#) : Represents file relationships including imports and being imported by.

- [WikiPage](#) : Represents a wiki page with content and metadata.
- [ChunkType](#) : Defines types of code chunks.
- [CodeChunk](#) : Represents a code chunk with metadata.
- [Language](#) : Defines programming languages.
- [build_file_to_wiki_map](#) : Builds a mapping from file paths to wiki page paths.

API Reference

class `TestRelationshipAnalyzer`

Tests for [RelationshipAnalyzer](#) class.

Methods:

`test_analyze_python_imports`

```
def test_analyze_python_imports()
```

Test analyzing Python import statements.

`test_get_relationships_imports`

```
def test_get_relationships_imports()
```

Test getting import relationships for a file.

`test_get_relationships_imported_by`

```
def test_get_relationships_imported_by()
```

Test finding files that import a given file.

`test_ignores_non_import_chunks`

```
def test_ignores_non_import_chunks()
```

Test that non-import chunks are ignored.

test_shared_dependencies

```
def test_shared_dependencies()
```

Test finding files with shared dependencies.

class TestBuildFileToWikiMap

Tests for [build_file_to_wiki_map](#) function.

Methods:**test_builds_correct_mapping**

```
def test_builds_correct_mapping()
```

Test that file paths are correctly mapped to wiki paths.

class TestGenerateSeeAlsoSection

Tests for [generate_see_also_section](#) function.

Methods:**test_generates_section_with_importers**

```
def test_generates_section_with_importers()
```

Test generating See Also with files that import this file.

test_generates_section_with_dependencies

```
def test_generates_section_with_dependencies()
```

Test generating See Also with dependency files.

test_returns_none_for_no_relationships

```
def test_returns_none_for_no_relationships()
```

Test that None is returned when no related pages exist.

test_avoids_self_reference

```
def test_avoids_self_reference()
```

Test that See Also doesn't include the current page.

class TestRelativePath

Tests for _relative_path function.

Methods:**test_same_directory**

```
def test_same_directory()
```

Test relative path in same directory.

test_parent_directory

```
def test_parent_directory()
```

Test relative path to parent directory.

test_sibling_directory

```
def test_sibling_directory()
```

Test relative path to sibling directory.

class TestAddSeeAlsoSections

Tests for [add_see_also_sections](#) function.

Methods:

test_adds_sections_to_file_pages

```
def test_adds_sections_to_file_pages()
```

Test that See Also sections are added to file documentation pages.

test_skips_non_file_pages

```
def test_skips_non_file_pages()
```

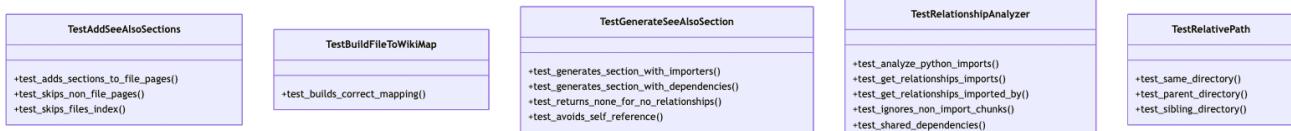
Test that non-file pages are not modified.

test_skips_files_index

```
def test_skips_files_index()
```

Test that files/index.md is not modified.

Class Diagram



Call Graph



Relevant Source Files

- tests/test_see_also.py

See Also

- [see_also](#) - dependency
- [models](#) - dependency
- [test_chunker](#) - shares 2 dependencies
- [test_api_docs](#) - shares 2 dependencies

File Overview

This file contains test cases for the source references functionality in the `local_deepwiki` project. It tests various aspects of how source file references are added to wiki pages, including handling of file-to-wiki mappings, line information, and section insertion logic.

Classes

TestBuildFileToWikiMap

Tests for the `build_file_to_wiki_map` function.

TestRelativePath

Tests for the `_relative_path` function.

Methods

- `test_same_directory` : Test relative path in same directory.
- `test_parent_directory` : Test relative path to parent directory.
- `test_child_directory` : Test relative path to child directory.

TestGenerateSourceRefsSection

Tests for `generate_source_refs_section` function.

Methods

- `test_single_file_with_wiki_link` : Test generating section for single file with wiki page.
- `test_single_file_without_wiki_link` : Test generating section for single file without wiki page.

- `test_multiple_files_with_wiki_links` : Test generating section for multiple files with wiki pages.
- `test_multiple_files_mixed_wiki_links` : Test generating section for multiple files with mixed wiki page availability.

TestAddSourceRefsSections

Tests for `add_source_refs_sections` function.

Methods

- `test_adds_sections_to_file_pages` : Test that sections are added to file documentation pages.
- `test_skips_index_pages` : Test that sections are not added to index pages.
- `test_inserts_before_see_also` : Test that sections are inserted before "See Also" section.
- `test_handles_missing_status` : Test handling of missing page status.
- `test_adds_section_to_module_pages` : Test that sections are added to module pages.
- `test_adds_section_to_architecture_page` : Test that sections are added to architecture page.

TestFormatFileEntry

Tests for `_format_file_entry` function.

Methods

- `test_without_line_info` : Test formatting without line info.
- `test_with_line_info` : Test formatting with line info shows start-end range.

TestGenerateSourceRefsSectionWithLineInfo

Tests for `generate_source_refs_section` with line info.

Methods

- `test_single_file_with_line_info` : Test single file displays line numbers.
- `test_multiple_files_with_line_info` : Test multiple files display line numbers.

TestAddSourceRefsSectionsWithLineInfo

Tests for `add_source_refs_sections` with line info in status.

Methods

- `test_passes_line_info_to_section_generator` : Test that line info from status is used in generated section.

Functions

build_file_to_wiki_map

Maps source file paths to corresponding wiki page paths.

generate_source_refs_section

Generates a section listing source files with optional wiki links and line information.

Parameters

- `source_files` (list of str): List of source file paths.
- `current_wiki_path` (str): Path of the current wiki page.
- `file_to_wiki` (dict): Mapping from source file paths to wiki page paths.
- `file_line_info` (dict, optional): Mapping from source file paths to line information.

Returns

- str: Formatted section content.

_format_file_entry

Formats a single file entry with optional wiki link and line information.

Parameters

- `file_path` (str): Path to the source file.
- `wiki_path` (str or None): Wiki page path for the file, or None if not available.
- `current_wiki_path` (str): Path of the current wiki page.
- `line_info` (dict or None): Line information for the file.

Returns

- str: Formatted file entry.

_relative_path

Calculates the relative path from one wiki page to another.

Parameters

- `from_path` (str): Source wiki page path.
- `to_path` (str): Target wiki page path.

Returns

- str: Relative path from `from_path` to `to_path`.

Usage Examples

Using `generate_source_refs_section`

```
result = generate_source_refs_section(
    source_files=["src/parser.py"],
    current_wiki_path="files/src/chunker.md",
    file_to_wiki={"src/parser.py": "files/src/parser.md"},
)
```

Using `add_source_refs_sections`

```
pages = [
    WikiPage(
        path="files/src/parser.md",
        title="parser",
        content="# Parser\n\nContent here.",
        generated_at=0,
    ),
]

page_statuses = {
    "files/src/parser.md": WikiPageStatus(
        path="files/src/parser.md",
        source_files=["src/parser.py"],
        source_hashes={},
        content_hash="xyz",
        generated_at=0,
    ),
}

result = add_source_refs_sections(pages, page_statuses)
```

Related Components

This file works with the following components:

- `WikiPage` from `local_deepwiki.models`
- `WikiPageStatus` from `local_deepwiki.models`
- `add_source_refs_sections` from `local_deepwiki.generators.source_refs`
- `build_file_to_wiki_map` from `local_deepwiki.generators.source_refs`
- `generate_source_refs_section` from `local_deepwiki.generators.source_refs`
- `_format_file_entry` from `local_deepwiki.generators.source_refs`
- `_relative_path` from `local_deepwiki.generators.source_refs`

API Reference

class `TestBuildFileToWikiMap`

Tests for `build_file_to_wiki_map` function.

Methods:

`test_builds_correct_mapping`

```
def test_builds_correct_mapping()
```

Test that file paths are correctly mapped to wiki paths.

`test_empty_pages`

```
def test_empty_pages()
```

Test with empty pages list.

class `TestRelativePath`

Tests for `_relative_path` function.

Methods:**test_same_directory**

```
def test_same_directory()
```

Test relative path in same directory.

test_parent_directory

```
def test_parent_directory()
```

Test relative path to parent directory.

test_sibling_directory

```
def test_sibling_directory()
```

Test relative path to sibling directory.

test_root_to_nested

```
def test_root_to_nested()
```

Test relative path from root to nested.

class TestGenerateSourceRefsSection

Tests for generate_source_refs_section function.

Methods:**test_single_file_with_wiki_link**

```
def test_single_file_with_wiki_link()
```

Test generating section for single file with wiki page.

test_single_file_no_wiki_page

```
def test_single_file_no_wiki_page()
```

Test generating section for file without wiki page.

test_multiple_files

```
def test_multiple_files()
```

Test generating section for multiple files.

test_empty_source_files

```
def test_empty_source_files()
```

Test that empty source files returns None.

test_max_items_limit

```
def test_max_items_limit()
```

Test that max_items limits the output.

test_skips_self_reference

```
def test_skips_self_reference()
```

Test that current page is not linked to itself.

class TestAddSourceRefsSections

Tests for add_source_refs_sections function.

Methods:**test_adds_sections_to_file_pages**

```
def test_adds_sections_to_file_pages()
```

Test that sections are added to file documentation pages.

test_skips_index_pages

```
def test_skips_index_pages()
```

Test that index pages are not modified.

test_inserts_before_see_also

```
def test_inserts_before_see_also()
```

Test that section is inserted before See Also.

test_handles_missing_status

```
def test_handles_missing_status()
```

Test that pages without status are passed through.

test_adds_section_to_module_pages

```
def test_adds_section_to_module_pages()
```

Test that sections are added to module pages.

test_adds_section_to_architecture_page

```
def test_adds_section_to_architecture_page()
```

Test that sections are added to architecture page.

class TestFormatFileEntry

Tests for _format_file_entry function.

Methods:

test_without_line_info

```
def test_without_line_info()
```

Test formatting without line info.

test_with_line_info

```
def test_with_line_info()
```

Test formatting with line info shows start-end range.

test_with_line_info_and_wiki_link

```
def test_with_line_info_and_wiki_link()
```

Test formatting with line info and wiki link.

test_skips_self_link_with_line_info

```
def test_skips_self_link_with_line_info()
```

Test that self-reference doesn't include link even with line info.

class TestGenerateSourceRefsSectionWithLineInfo

Tests for generate_source_refs_section with line info.

Methods:**test_single_file_with_line_info**

```
def test_single_file_with_line_info()
```

Test single file displays line numbers.

test_multiple_files_with_line_info

```
def test_multiple_files_with_line_info()
```

Test multiple files each display their line numbers.

`test_partial_line_info`

```
def test_partial_line_info()
```

Test that files without line info fallback gracefully.

`class TestAddSourceRefsSectionsWithLineInfo`

Tests for add_source_refs_sections with line info in status.

Methods:

`test_passes_line_info_to_section_generator`

```
def test_passes_line_info_to_section_generator()
```

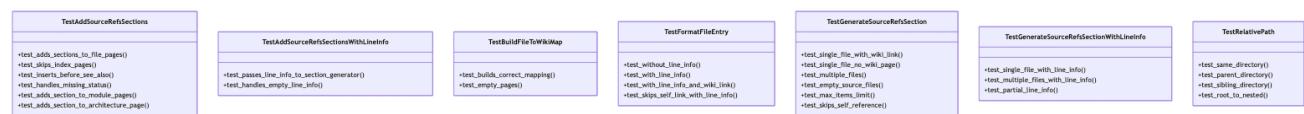
Test that line info from status is used in generated section.

`test_handles_empty_line_info`

```
def test_handles_empty_line_info()
```

Test that empty line info works (fallback to no line numbers).

Class Diagram



Call Graph



Relevant Source Files

- `tests/test_source_refs.py:15–51`

See Also

- [test_diagrams](#) - shares 2 dependencies

File Overview

This file contains unit tests for the `DebouncedHandler` and `RepositoryWatcher` classes from the `local_deepwiki.watcher` module. The tests verify the functionality of file watching, debouncing, and reindex scheduling for supported file extensions.

Classes

TestWatchedExtensions

Tests the `WATCHED_EXTENSIONS` constant to ensure that the correct file extensions are included for monitoring.

Methods

- `test_python_extensions` : Verifies that Python file extensions (`.py`, `.pyi`) are included in `WATCHED_EXTENSIONS`.
- `test_javascript_extensions` : Verifies that JavaScript and TypeScript file extensions (`.js`, `.jsx`, `.ts`, `.tsx`) are included in `WATCHED_EXTENSIONS`.
- `test_other_extensions` : Verifies that other file extensions (`.md`, `.txt`, `.yml`, `.yaml`, `.json`) are included in `WATCHED_EXTENSIONS`.

TestDebouncedHandler

Tests the `DebouncedHandler` class, which handles file system events and schedules reindexing after a debounce period.

Methods

- `test_should_watch_python_file` : Verifies that Python files are correctly identified as watchable.
- `test_on_modified_schedules_reindex` : Verifies that file modifications schedule a reindex.
- `test_on_created_schedules_reindex` : Verifies that file creation schedules a reindex.

- `test_on_deleted_schedules_reindex` : Verifies that file deletion schedules a reindex.
- `test_directory_events_ignored` : Verifies that directory events are ignored.
- `test_non_watched_file_ignored` : Verifies that non-watched file types are ignored.
- `test_multiple_changes_debounced` : Verifies that multiple rapid changes are debounced into a single reindex.

TestRepositoryWatcher

Tests the `RepositoryWatcher` class, which manages a file system watcher for a repository.

Methods

- `test_create_watcher` : Verifies that a `RepositoryWatcher` can be created with default settings.
- `test_create_watcher_with_options` : Verifies that a `RepositoryWatcher` can be created with custom configuration and debounce settings.

TestDebouncedHandlerEvents

A class containing tests for events handled by `DebouncedHandler`. This class includes methods for testing file modification, creation, deletion, and other event types.

Methods

- `handler_with_mock` : A pytest fixture that provides a mocked `DebouncedHandler` instance.
- `test_on_modified_schedules_reindex` : Tests that file modifications schedule a reindex.
- `test_on_created_schedules_reindex` : Tests that file creation schedules a reindex.
- `test_on_deleted_schedules_reindex` : Tests that file deletion schedules a reindex.
- `test_directory_events_ignored` : Tests that directory events are ignored.
- `test_non_watched_file_ignored` : Tests that non-watched file types are ignored.
- `test_multiple_changes_debounced` : Tests that multiple rapid changes are debounced.

Functions

No standalone functions are defined in this file. All functionality is encapsulated in test methods within the classes listed above.

Usage Examples

The tests use pytest fixtures and mocking to isolate and test `DebouncedHandler` behavior. For example:

```
def test_on_modified_schedules_reindex(self, handler_with_mock, tmp_path):
    test_file = tmp_path / "test.py"
    test_file.touch()

    event = MagicMock()
    event.is_directory = False
    event.src_path = str(test_file)

    handler_with_mock.on_modified(event)

    assert str(test_file) in handler_with_mock._pending_files
```

Related Components

This file works with the following components from the `local_deepwiki.watcher` module:

- `DebouncedHandler` : Handles file system events and schedules reindexing.
- `RepositoryWatcher` : Manages a file system watcher for a repository.
- `WATCHED_EXTENSIONS` : A set of file extensions that are monitored for changes.

It also uses:

- `Config` : Configuration class for the application.
- `MagicMock` : For mocking objects in tests.
- `pytest` : Testing framework.
- `time` : For time-related operations in tests.

- `pathlib.Path` : For path manipulation in tests.

API Reference

class `TestWatchedExtensions`

Test that watched extensions are correct.

Methods:

`test_python_extensions`

```
def test_python_extensions()
```

Test Python extensions are watched.

`test_javascript_extensions`

```
def test_javascript_extensions()
```

Test JavaScript/TypeScript extensions are watched.

`test_other_extensions`

```
def test_other_extensions()
```

Test other language extensions are watched.

class `TestDebouncedHandler`

Test [DebouncedHandler](#) functionality.

Methods:

`handler`

```
def handler(tmp_path)
```

Create a handler for testing.

Parameter	Type	Default	Description
tmp_path	-	-	-

`test_should_watch_python_file`

```
def test_should_watch_python_file(handler, tmp_path)
```

Test that Python files are watched.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

`test_should_watch_typescript_file`

```
def test_should_watch_typescript_file(handler, tmp_path)
```

Test that TypeScript files are watched.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

`test_should_not_watch_text_file`

```
def test_should_not_watch_text_file(handler, tmp_path)
```

Test that text files are not watched.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

test_should_not_watch_json_file

```
def test_should_not_watch_json_file(handler, tmp_path)
```

Test that JSON files are not watched.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

test_should_exclude_node_modules

```
def test_should_exclude_node_modules(handler, tmp_path)
```

Test that node_modules files are excluded.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

test_should_exclude_venv

```
def test_should_exclude_venv(handler, tmp_path)
```

Test that venv files are excluded.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

test_should_exclude_pycache

```
def test_should_exclude_pycache(handler, tmp_path)
```

Test that **pycache** files are excluded.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

test_should_exclude_git

```
def test_should_exclude_git(handler, tmp_path)
```

Test that .git files are excluded.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

test_should_watch_nested_file

```
def test_should_watch_nested_file(handler, tmp_path)
```

Test that nested source files are watched.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

test_file_outside_repo_not_watched

```
def test_file_outside_repo_not_watched(handler, tmp_path)
```

Test that files outside repo are not watched.

Parameter	Type	Default	Description
handler	-	-	-
tmp_path	-	-	-

class TestRepositoryWatcher

Test [RepositoryWatcher](#) functionality.

Methods:**test_create_watcher**

```
def test_create_watcher(tmp_path)
```

Test creating a watcher.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_create_watcher_with_options

```
def test_create_watcher_with_options(tmp_path)
```

Test creating a watcher with options.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_start_stop_watcher

```
def test_start_stop_watcher(tmp_path)
```

Test starting and stopping a watcher.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_stop_without_start

```
def test_stop_without_start(tmp_path)
```

Test stopping a watcher that was never started.

Parameter	Type	Default	Description
tmp_path	-	-	-

class TestDebouncedHandlerEvents

Test event handling with debouncing.

Methods:

handler_with_mock

```
def handler_with_mock(tmp_path)
```

Create a handler with mocked reindex.

Parameter	Type	Default	Description
tmp_path	-	-	-

test_on_modified_schedules_reindex

```
def test_on_modified_schedules_reindex(handler_with_mock, tmp_path)
```

Test that file modification schedules reindex.

Parameter	Type	Default	Description
handler_with_mock	-	-	-
tmp_path	-	-	-

test_on_created_schedules_reindex

```
def test_on_created_schedules_reindex(handler_with_mock, tmp_path)
```

Test that file creation schedules reindex.

Parameter	Type	Default	Description
handler_with_mock	-	-	-
tmp_path	-	-	-

test_on_deleted_schedules_reindex

```
def test_on_deleted_schedules_reindex(handler_with_mock, tmp_path)
```

Test that file deletion schedules reindex.

Parameter	Type	Default	Description
handler_with_mock	-	-	-
tmp_path	-	-	-

test_directory_events_ignored

```
def test_directory_events_ignored(handler_with_mock, tmp_path)
```

Test that directory events are ignored.

Parameter	Type	Default	Description
handler_with_mock	-	-	-
tmp_path	-	-	-

test_non_watched_file_ignored

```
def test_non_watched_file_ignored(handler_with_mock, tmp_path)
```

Test that non-watched files are ignored.

Parameter	Type	Default	Description
handler_with_mock	-	-	-
tmp_path	-	-	-

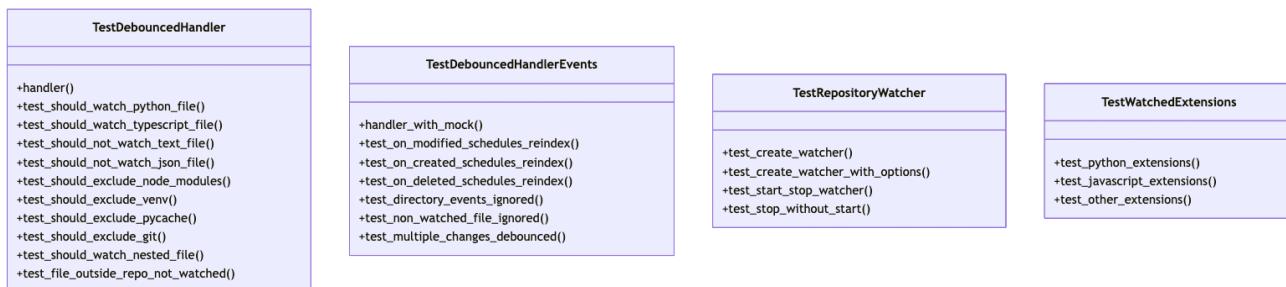
test_multiple_changes_debounced

```
def test_multiple_changes_debounced(handler_with_mock, tmp_path)
```

Test that multiple rapid changes are debounced.

Parameter	Type	Default	Description
handler_with_mock	-	-	-
tmp_path	-	-	-

Class Diagram



Call Graph



Relevant Source Files

- `tests/test_watcher.py:17–39`

Test Web Documentation

File Overview

This file contains unit tests for the web application components of `local_deepwiki`. It tests the Flask application initialization and breadcrumb building functionality, ensuring proper behavior for wiki navigation.

Classes

`TestBuildBreadcrumb`

Tests for the `build_breadcrumb` function that generates navigation breadcrumbs for wiki pages.

Methods:

- `test_root_page_noBreadcrumb(wiki_dir)`
 - Tests that root pages (index.md, architecture.md) return empty breadcrumbs
 - Parameters: `wiki_dir` - temporary directory path for test wiki
 - Asserts: Empty string result for root pages
- `test_simple_nested_page(wiki_dir)`
 - Tests breadcrumb generation for pages one level deep
 - Parameters: `wiki_dir` - temporary directory path for test wiki
 - Asserts: Breadcrumb contains Home link and proper page hierarchy

`TestFlaskApp`

Tests for Flask application functionality including initialization and routing.

Methods:

- `test_create_app(wiki_dir)`

- Tests that `create_app` initializes correctly with a valid wiki directory
- Parameters: `wiki_dir` - temporary directory path for test wiki
- Asserts: App instance is returned (not None)
- `test_create_app_invalid_path()`
- Tests that `create_app` raises ValueError for non-existent paths
- Asserts: ValueError with "does not exist" message
- `test_index_redirect(wiki_dir)`
- Tests that the root URL (/) redirects to /wiki/index.md
- Parameters: `wiki_dir` - temporary directory path for test wiki
- Asserts: HTTP 302 redirect response

Functions

`wiki_dir`

Purpose: A pytest fixture that creates a temporary directory structure for testing wiki content.

Parameters: None

Returns: Path object pointing to temporary directory containing test wiki files

Usage:

```
def test_something(wiki_dir):
    # wiki_dir contains test wiki structure
    pass
```

Usage Examples

Running Tests

```
pytest tests/test_web.py
```

Testing App Creation

```
def test_app_initialization(wiki_dir):
    app = create_app(wiki_dir)
    assert app is not None
```

Testing Breadcrumb Generation

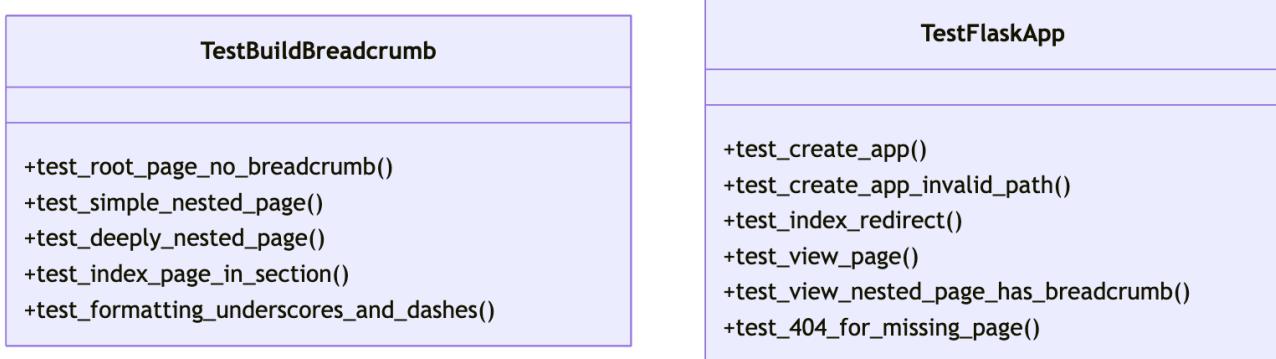
```
def test_breadcrumb_generation(wiki_dir):
    breadcrumb = buildBreadcrumb(wiki_dir, "modules/src.md")
    assert '<a href=' in breadcrumb # Contains Home link
```

Dependencies

- `pytest` - Testing framework
- `pathlib.Path` - Path manipulation
- `tempfile` - Temporary file handling
- `local_deepwiki.web.app` - Contains `buildBreadcrumb` and `create_app` functions being tested

The test file requires the `local_deepwiki` package to be installed and accessible in the Python path.

Class Diagram



See Also

- [app](#) - dependency
- [test_search](#) - shares 4 dependencies
- [test_parser](#) - shares 3 dependencies
- [test_chunker](#) - shares 2 dependencies