File Breakdown: `src/utils/function_logger.py`

File Location

`src/utils/function_logger.py`

Overview

The `function_logger.py` file implements a simple function logging decorator that enables tracking and monitoring of function calls throughout the multi-agent system. This lightweight utility helps with debugging, auditing, and understanding the flow of execution across different components.

Key Responsibilities

- Log function calls to the console
- Display function module and name information
- Track the execution flow across the system
- Provide a consistent logging mechanism for all functions
- Support debugging and monitoring activities

Core Functionality

Decorator Definition

The entire file consists of a single decorator function:

```python
def log_function_call(func):
    """Decorator to log function calls."""
    def wrapper(*args, **kwargs):
        print(f"\nFUNCTION CALL: {func.__module__}.{func.__name__}")
        return func(*args, **kwargs)
    return wrapper
```

This decorator wraps any function it's applied to, printing a log message with the function's module name and function name before calling the original function.

Key Features

1. **Simplicity**: A minimal implementation that serves a crucial purpose
2. **Non-Intrusive**: Doesn't change the function's behavior or return value
3. **Module Context**: Includes the module name for proper context
4. **Visibility**: Clearly denotes function boundaries in console output
5. **Performance**: Minimal overhead for logging operations

Integration

- Applied to functions across the entire system using the `@log_function_call` syntax
- Used in agent methods, utility functions, and core operations
- Provides consistent logging format throughout the application

- Used extensively with both synchronous and asynchronous functions

Usage Examples

The decorator is applied to most functions in the system:

```
@log_function_call
def do_maths(kb: KnowledgeBase, prompt: str, input2: str = "-") -> str:
    # Function implementation...

@log_function_call
async def handle_task_async(self, task: Task):
    # Async function implementation...

@log_function_call
def process_emil_request(kb: KnowledgeBase, prompt: str = None, **kwargs):
    # Function implementation...
```

Workflow

1. When a decorated function is called, the wrapper function executes first
2. The wrapper prints a log message with the module and function name
3. The original function is called with all arguments passed through
4. The return value from the original function is passed back to the caller

Implementation Notes

- Uses Python's decorator pattern for clean, non-invasive logging
- Relies on the `__module__` and `__name__` attributes of function objects
- Preserves the original function's signature and behavior
- Could be extended to include more information (timestamps, argument values, etc.)
- Helps track the flow of execution across asynchronous and synchronous operations