```
import json
from typing import List, Any, Callable

from swarms.utils.parse_code import extract_code_from_markdown
from swarms.utils.loguru_logger import initialize_logger

logger = initialize_logger(log_folder="tool_parse_exec")


def parse_and_execute_json(
    functions: List[Callable[..., Any]],
    json_string: str,
    parse_md: bool = False,
    verbose: bool = False,
    return_str: bool = True,
) -> dict:
    """
    Parses and executes a JSON string containing function names and parameters.

    Args:
        functions (List[callable]): A list of callable functions.
        json_string (str): The JSON string to parse and execute.
        parse_md (bool): Flag indicating whether to extract code from Markdown.
        verbose (bool): Flag indicating whether to enable verbose logging.
        return_str (bool): Flag indicating whether to return a JSON string.
    Returns:
```

```
    dict: A dictionary containing the results of executing the functions with the parsed parameters.
"""
if not functions or not json_string:
    raise ValueError("Functions and JSON string are required")

if parse_md:
    json_string = extract_code_from_markdown(json_string)

try:
    # Create function name to function mapping
    function_dict = {func.__name__: func for func in functions}

    if verbose:
        logger.info(
            f"Available functions: {list(function_dict.keys())}"
        )
        logger.info(f"Processing JSON: {json_string}")

    # Parse JSON data
    data = json.loads(json_string)

    # Handle both single function and function list formats
    function_list = []
    if "functions" in data:
        function_list = data["functions"]
    elif "function" in data:
```

```
    function_list = [data["function"]]
else:
    function_list = [
        data
    ]  # Assume entire object is single function

# Ensure function_list is a list and filter None values
if isinstance(function_list, dict):
    function_list = [function_list]
function_list = [f for f in function_list if f]

if verbose:
    logger.info(f"Processing {len(function_list)} functions")

results = {}
for function_data in function_list:
    function_name = function_data.get("name")
    parameters = function_data.get("parameters", {})

    if not function_name:
        logger.warning("Function data missing name field")
        continue

    if verbose:
        logger.info(
            f"Executing {function_name} with params: {parameters}"
```

```
            )

        if function_name not in function_dict:
            logger.warning(f"Function {function_name} not found")
            results[function_name] = None
            continue

        try:
            result = function_dict[function_name](**parameters)
            results[function_name] = str(result)
            if verbose:
                logger.info(
                    f"Result for {function_name}: {result}"
                )
        except Exception as e:
            logger.error(
                f"Error executing {function_name}: {str(e)}"
            )
            results[function_name] = f"Error: {str(e)}"

    # Format final results
    if len(results) == 1:
        # Return single result directly
        data = {"result": next(iter(results.values()))}
    else:
        # Return all results
```

```
            data = {
                "results": results,
                "summary": "\n".join(
                    f"{k}: {v}" for k, v in results.items()
                ),
            }

        if return_str:
            return json.dumps(data)
        else:
            return data

    except json.JSONDecodeError as e:
        error = f"Invalid JSON format: {str(e)}"
        logger.error(error)
        return {"error": error}
    except Exception as e:
        error = f"Error parsing and executing JSON: {str(e)}"
        logger.error(error)
        return {"error": error}
```