# Market Research & Use Case Generation Agent - Source Code Documentation

**1. Overview**

The multi-agent system is designed to automate market research, use case generation, and resource collection for AI/ML applications. It comprises several distinct modules that work together to retrieve, analyze, and present data to the user.

**2. Code Architecture**

The project is structured into four main components:

1. **User Interface (UI)**
2. **Multi-Agent System**
3. **External API Interfaces**
4. **Output Formatting**

---

**3. Key Components**

**A. User Interface (UI)**

- **app.py**
    - **Role**: Acts as the entry point for the Streamlit-based front end. It provides fields for user input, initiates backend agents, and displays the results.
    - **Key Interactions**:
        - Accepts user input (e.g., company name, industry).
        - Invokes each agent and presents their outputs.
        - Manages error handling to ensure the user is informed of any issues during execution.
    - **Primary Functions**:
        - main(): The primary function coordinating the application's flow.
        - display_results(): Shows results from each agent on the UI.

**B. Multi-Agent System**

1. **Market Research Agent (market_research_agent.py)**
    - **Role**: Collects information about the industry and the target company by querying external web sources and APIs.
    - **Key Interactions**:

        - Sends HTTP requests to a web search API to fetch data.
        - Filters and structures data into JSON format for easy processing.
    - **Primary Functions**:
        - fetch_industry_info(company): Calls an API to retrieve recent articles and insights.
        - parse_results(data): Parses and structures results into a format usable by downstream components.
2. **Use Case Generation Agent (use_case_generator.py)**
    - **Role**: Based on market research insights, generates AI/ML use cases aligned with the company's goals.
    - **Key Interactions**:
        - Analyzes JSON data from the Market Research Agent to identify relevant use cases.
        - Uses pre-defined templates or heuristics to generate use cases in line with industry trends.
    - **Primary Functions**:
        - generate_use_cases(industry_info, focus_areas): Processes information and outputs a list of potential AI/ML use cases.
3. **Resource Asset Collection Agent (resource_collector.py)**
    - **Role**: Finds datasets and resources relevant to the generated use cases.
    - **Key Interactions**:
        - Queries external resources like Kaggle or Hugging Face based on the generated use cases.
        - Formats resource links in Markdown format.
    - **Primary Functions**:
        - collect_resources(use_cases): Queries and retrieves relevant datasets and resource links.
        - format_as_markdown(links): Formats links into a user-friendly markdown file.

**C. External API Interfaces**

- **API Interface (api_interface.py)**
    - **Role**: Facilitates communication with external APIs such as Serper, Hugging Face, and Kaggle.
    - **Key Interactions**:
        - Sends requests and retrieves responses from the APIs.
        - Manages authentication, error handling, and response parsing.
    - **Primary Functions**:
        - search_web(query): Calls a web search API with a specific query.
        - fetch_datasets(query): Fetches relevant datasets based on provided keywords.

**D. Output Formatting**

- **Output Formatter (output_formatter.py)**
    - **Role**: Organizes the results from the agents into a presentable format, particularly useful for the UI.
    - **Key Interactions**:
        - Accepts data from agents and composes a report-style output.
    - **Primary Functions**:
        - format_for_display(data): Converts JSON data into a structured HTML format suitable for Streamlit.
        - export_markdown(results): Saves the output as a downloadable Markdown file.

---

**4. Code Flow and Interactions**

1. **User Interaction**:
    - The user provides input through the Streamlit UI (app.py).
2. **Market Research**:
    - The Market Research Agent fetches industry and company-specific data, storing it in JSON format for further processing.
3. **Use Case Generation**:
    - The Use Case Generation Agent takes research data and generates relevant AI/ML use cases based on the company's focus areas.
4. **Resource Collection**:
    - The Resource Asset Collection Agent searches for datasets and resources related to each use case and structures these links in Markdown format.
5. **Display and Output**:
    - The Output Formatter organizes all data into a report-style layout, which is then displayed on the UI. Users can also download the resource links as a Markdown file.

---

**5. Error Handling**

- **KeyError Handling**: The system checks for missing fields in the data (e.g., 'industry_information'), and displays a warning on the UI if required data is missing.
- **API Call Errors**: The API Interface module logs errors when API calls fail and informs the user through the Streamlit app.

---

**6. Dependencies**

- **External Libraries**:
    - Streamlit: For building the interactive UI.
    - Requests: For making HTTP requests to external APIs.
    - json: For handling JSON data parsing and formatting.
- **API Keys**: Store keys in environment variables for security and load them when needed.

---

## 7. Main Function (`app.py`)

```
python
Copy code
import streamlit as st
from market_research_agent import fetch_industry_info
from use_case_generator import generate_use_cases
from resource_collector import collect_resources

def main():
    st.title("Market Research & Use Case Generation")

    # Input form
    company = st.text_input("Enter the company name")
    if st.button("Generate"):
        # Step 1: Market Research
        research_data = fetch_industry_info(company)

        # Step 2: Generate Use Cases
        use_cases = generate_use_cases(research_data['industry_info'],
research_data['focus_areas'])

        # Step 3: Collect Resources
        resources = collect_resources(use_cases)

        # Display results
        st.write("Generated Use Cases:")
        st.write(use_cases)

        st.write("Resources:")
        st.write(resources)

if __name__ == "__main__":
    main()
```