Complete Source Code Documentation

# 1. Project Overview

The Multi-Agent Architecture is a system designed to automate the process of researching companies, identifying market trends, and generating AI-driven use cases, with integration to Kaggle for resource collection. The system is divided into three primary agents:  
1. Agent 1: Browser Agent for Company Research  
2. Agent 2: Market Standards & Use Case Generation  
3. Agent 3: Resource Asset Collection  
Each agent interacts with the backend to fetch, analyze, and present data via a Streamlit-based frontend.

# 2. File Structure

project/  
├── app.py # Streamlit app to interact with users  
├── agents/  
│ ├── browser\_agent.py # Agent 1: Web scraping and company research  
│ ├── market\_agent.py # Agent 2: Market trends and use case generation  
│ └── resource\_agent.py # Agent 3: Kaggle resource collection  
├── utils/  
│ ├── scraper.py # Utility functions for web scraping  
│ ├── classifier.py # Utility functions for industry classification  
│ └── data\_storage.py # Interface with Qdrant database  
├── requirements.txt # Project dependencies  
└── README.md # Project overview and setup instructions

# 3. Detailed Code Documentation

## 3.1 app.py

This file serves as the front-end user interface using Streamlit. It handles user input, displays results, and interacts with the backend agents.  
  
import streamlit as st  
from agents.browser\_agent import BrowserAgent  
from agents.market\_agent import MarketAgent  
from agents.resource\_agent import ResourceAgent  
def run\_app():  
 st.title('Multi-Agent Market Research System')  
 company\_name = st.text\_input('Enter Company Name')  
 if st.button('Run Analysis'):  
 browser\_agent = BrowserAgent(company\_name)  
 company\_data = browser\_agent.run\_research()  
 market\_agent = MarketAgent(company\_data['industry'])  
 use\_cases = market\_agent.generate\_use\_cases()  
 resource\_agent = ResourceAgent(company\_data['industry'])  
 resources = resource\_agent.collect\_resources()  
 st.write('Company Data:', company\_data)  
 st.write('Use Cases:', use\_cases)  
 st.write('Resources:', resources)  
if \_\_name\_\_ == '\_\_main\_\_':  
 run\_app()

## 3.2 agents/browser\_agent.py

This agent is responsible for researching the company, scraping relevant information from the web, and classifying the industry.  
  
import requests  
from utils.scraper import extract\_relevant\_text  
from utils.classifier import classify\_industry  
class BrowserAgent:  
 def \_\_init\_\_(self, company\_name):  
 self.company\_name = company\_name  
 def run\_research(self):  
 search\_url = f'https://www.google.com/search?q={self.company\_name}'  
 response = requests.get(search\_url)  
 company\_info = extract\_relevant\_text(response.text)  
 industry = classify\_industry(company\_info)  
 return {'company\_name': self.company\_name, 'industry': industry, 'content': company\_info}

## 3.3 agents/market\_agent.py

This agent uses the company’s industry information to propose relevant use cases and trends.  
  
class MarketAgent:  
 def \_\_init\_\_(self, industry):  
 self.industry = industry  
 def generate\_use\_cases(self):  
 use\_cases = {  
 'Automotive': ['AI-powered autonomous driving', 'Predictive maintenance using IoT'],  
 'Finance': ['AI-driven fraud detection', 'Customer behavior analysis with machine learning']  
 }  
 return use\_cases.get(self.industry, ['No use cases available'])

## 3.4 agents/resource\_agent.py

This agent collects Kaggle datasets related to the company’s industry and provides them to the user.  
  
import kaggle  
class ResourceAgent:  
 def \_\_init\_\_(self, industry):  
 self.industry = industry  
 def collect\_resources(self):  
 resources = kaggle.api.datasets\_list(search=self.industry)  
 resource\_links = [f'https://www.kaggle.com/datasets/{resource.ref}' for resource in resources]  
 return resource\_links

## 3.5 utils/scraper.py

Utility functions for web scraping.  
  
from bs4 import BeautifulSoup  
def extract\_relevant\_text(html\_content):  
 soup = BeautifulSoup(html\_content, 'html.parser')  
 paragraphs = soup.find\_all('p')  
 return ' '.join([para.get\_text() for para in paragraphs])

## 3.6 utils/classifier.py

Utility functions for classifying the company’s industry.  
  
industry\_keywords = {  
 'Automotive': ['vehicle', 'car', 'automotive', 'engine', 'transport'],  
 'Finance': ['finance', 'banking', 'investment', 'capital', 'credit']  
}  
def classify\_industry(text):  
 for industry, keywords in industry\_keywords.items():  
 if any(keyword in text.lower() for keyword in keywords):  
 return industry  
 return 'Unknown'