AI Strategy Analysis Tool - Source Code Documentation

Code Architecture Overview

The AI Strategy Analysis Tool is structured around a core `MarketResearchAgent` class that handles all the analysis functionality, with a Streamlit-based user interface for interaction. The application follows a modular design pattern where each component has a specific responsibility in the analysis workflow.

Key Components and Their Roles

1. Main Module (`main.py`)

The main module serves as the entry point to the application and contains two primary components:

1. **MarketResearchAgent Class**: The core business logic component that handles all analysis operations

2. **Main Function**: Manages the Streamlit UI and orchestrates the analysis workflow

2. MarketResearchAgent Class

The `MarketResearchAgent` class is the central component responsible for all analysis operations. It encapsulates the following functionality:

```
class MarketResearchAgent:
    def __init__(self)
    def determine_industry(self, company_name)
    def research_company(self, company_name)
    def generate_use_cases(self, company_name, industry, research_insights)
    def collect_resource_assets(self, use_cases)
    def generate_final_proposal(self, company_name, industry, use_cases, resource_map,
research_insights)
```

2.1 Initialization (`__init__`)

```
def __init__(self):
    # Load API keys from Streamlit Secrets
    self.exa = Exa(api_key=os.getenv("EXA_API_KEY"))
    self.groq_client = groq.Client(api_key=os.getenv("GROQ_API_KEY"))
    self.llm_config = {
        "config_list": [{
            "model": "llama-3.1-8b-instant",
            "api_key": os.getenv("GROQ_API_KEY"),
            "api_type": "groq"
        }]
    }
```

This method initializes the API clients for Exa (web search) and Groq (LLM), and sets up the configuration for the LLM.

2.2 Industry Determination (`determine_industry`)

```
def determine_industry(self, company_name):
    """Determine the industry of a company using AI analysis"""
    industry_prompt = f"""
    Analyze the company {company_name} and determine its primary industry.
    Return only the industry name without any explanation or additional text.
    Example response format: "Electric Vehicles" or "Cloud Computing"
    """

    assistant = AssistantAgent("industry_analyzer", llm_config=self.llm_config)
    response = assistant.generate_reply(messages=[{"content": industry_prompt, "role": "user"}])

    industry = str(response).strip()

    return industry
```

2.3 Company Research (`research_company`)

```
def research_company(self, company_name):
    """Conduct comprehensive company and industry research"""
    industry = self.determine_industry(company_name)

    search_queries = [
        f"{industry} industry overview and trends",
        f"{company_name} strategic focus and market position",
        f"{industry} technological innovations and future outlook"
    ]
```

```
    research_results = {}
    for query in search_queries:
        try:
            results = self.exa.search(query, num_results=3, type="neural")
            research_results[query] = [
                {"title": res.title, "url": res.url, "snippet": res.summary}
                for res in results.results
            ]
        except Exception as e:
            st.error(f"Research error for query '{query}': {e}")

    return research_results, industry
```

2.4 Use Case Generation (`generate_use_cases`)

```
def generate_use_cases(self, company_name, industry, research_insights):
    """Generate AI/ML use cases based on company research"""
    use_case_prompt = f"""
    Based on the analysis of {company_name} in the {industry} industry, generate innovative
AI/ML use cases.
    Research insights: {json.dumps(research_insights)}
    Focus on:
    - Operational efficiency
    - Customer experience enhancement
    - Technological innovation
    Give at least 5 detailed distinct use cases.
    Avoid giving code or implementation details.
    Provide specific, actionable recommendations.
    """

    assistant = AssistantAgent("use_case_generator", llm_config=self.llm_config)
    use_cases = assistant.generate_reply(messages=[{"content": use_case_prompt, "role":
"user"}])

    return [case.strip() for case in re.split(r'\d+\.', str(use_cases)) if case.strip()]
```

2.5 Resource Collection (`collect_resource_assets`)

```
def collect_resource_assets(self, use_cases):
    """Collect dataset resources for each use case"""
    resource_map = {}
    platforms = [
        "kaggle.com/datasets",
        "huggingface.co/datasets",
```

```
        "github.com/datasets"
    ]

    for use_case in use_cases:
        resources = []
        for platform in platforms:
            query = f'"{use_case}" dataset site:{platform}'
            try:
                results = self.exa.search(query, num_results=3)
                platform_resources = [
                    {"url": res.url, "title": res.title}
                    for res in results.results
                    if platform in res.url and any(keyword in res.url.lower() for keyword in ['dataset',
'data'])
                ]
                resources.extend(platform_resources)
            except Exception as e:
                st.error(f"Resource search error for '{use_case}' on {platform}: {e}")

        resource_map[use_case] = resources

    return resource_map
```

2.6 Proposal Generation (`generate_final_proposal`)

```
def generate_final_proposal(self, company_name, industry, use_cases, resource_map,
research_insights):
    """Create comprehensive markdown proposal"""
    proposal = f"# AI Strategy Analysis for {company_name}\n\n"
    proposal += f"## Industry: {industry}\n\n"
    proposal += f"*Generated on {datetime.now().strftime('%B %d, %Y')}*\n\n"

    proposal += "## Market Research Insights\n"
    for query, insights in research_insights.items():
        proposal += f"### {query.title()}\n\n"
        for insight in insights:
            date_str = f" ({insight.get('published_date', 'N/A')})" if 'published_date' in insight and
insight.get('published_date', 'N/A') != 'N/A' else ""
            proposal += f"#### {insight['title']}{date_str}\n\n"
            proposal += f"{insight['snippet']}\n\n"
            proposal += f"[Read more]({insight['url']})\n\n"
            proposal += "---\n\n"

    proposal += "## Recommended AI/ML Use Cases\n\n"
```

```
    for idx, use_case in enumerate(use_cases, 1):
        proposal += f"### Use Case {idx}: {use_case}\n\n"

        if use_case in resource_map and resource_map[use_case]:
            proposal += "#### Recommended Datasets:\n"
            for resource in resource_map[use_case]:
                proposal += f"- [{resource['title']}]({resource['url']})\n"

        proposal += "\n"

    return proposal
```

3. Main Function

The `main()` function sets up the Streamlit UI and orchestrates the analysis workflow:

```
def main():
    st.set_page_config(
        page_title="AI Strategy Analysis",
        page_icon="favicon.ico",
    )

    # Custom CSS for modern UI
    st.markdown("""<style>
    .stApp { background-color: #111121; }
    .stApp .main { padding: 2rem; }
    .st-emotion-cache-16idsys p { font-size: 1.1rem; }
    .stButton > button {
        background-color: #9933FF;
        color: white;
        border: none;
    }
    .stButton > button:hover {
        background-color: #7F00FF;
        color: white;
    }
    </style>""", unsafe_allow_html=True)

    st.title("🧠 AI Strategy Analysis")
    st.markdown("""
    Generate comprehensive AI strategy proposals for any company.
    This tool analyzes a company, identifies AI opportunities, and creates a detailed
implementation plan.
```

```
    """)

    research_agent = MarketResearchAgent()
    company_name = st.text_input("Enter Company Name", help="Enter the company you want
to analyze")

    if st.button("Generate Analysis", type="primary"):
        if company_name:
            progress_placeholder = st.empty()
            progress_bar = st.progress(0)

            with st.spinner("Analyzing company data..."):
                progress_placeholder.markdown("🔍 **Phase 1:** Identifying industry and gathering
intelligence...")
                research_insights, industry = research_agent.research_company(company_name)
                st.sidebar.success(f"Industry Identified: {industry}")
                progress_bar.progress(25)

                progress_placeholder.markdown("💡 **Phase 2:** Identifying strategic
opportunities...")
                use_cases = research_agent.generate_use_cases(company_name, industry,
research_insights)
                progress_bar.progress(50)

                progress_placeholder.markdown("📚 **Phase 3:** Collecting supporting resources...")
                resource_map = research_agent.collect_resource_assets(use_cases)
                progress_bar.progress(75)

                progress_placeholder.markdown("📊 **Phase 4:** Generating comprehensive
report...")
                final_proposal = research_agent.generate_final_proposal(
                    company_name, industry, use_cases, resource_map, research_insights
                )
                progress_bar.progress(100)
                progress_placeholder.empty()

                # Display Results (Not included in snippet)
        else:
            st.warning("Please enter a company name to begin the analysis.")

    st.markdown("---")
    st.markdown("<div style='text-align: center; color: #666;'>"
                "<small>Powered by Groq and Exa | By Ayush Aditya | 2025 | AI Planet
Project</small></div>", unsafe_allow_html=True)
```

Component Interactions and Data Flow

1. **User Input** → **Main Function**: User inputs company name
2. **Main Function** → **MarketResearchAgent**: Methods called in sequence
3. **MarketResearchAgent** → **External APIs**: Groq & Exa for data/insights
4. **MarketResearchAgent** → **Main Function**: Returns structured data to UI

Data Structures

Research Insights

```
{
   "query1": [
      {"title": "Article Title", "url": "URL", "snippet": "Summary Text"},
   ],
   # More queries...
}
```

Use Cases

```
[
   "Use case 1 description",
   "Use case 2 description",
]
```

Resource Map

```
{
   "Use Case 1": [
      {"title": "Dataset Title", "url": "URL"},
   ]
}
```

Final Proposal

Markdown string with:

- Company/industry details
- Insights
- Use cases
- Datasets

Error Handling

- **Try-except** around API calls
- Fallbacks for industry extraction
- UI validations (e.g., company input)

Performance

- Search result limits
- Prompt token efficiency
- UI progress indicators

Security

- API keys via `.env` + `python-dotenv`
- No persistent data storage

Conclusion

This codebase is modular and well-structured, separating UI, logic, and external integration. The `MarketResearchAgent` ensures centralized orchestration, making the tool scalable and maintainable.