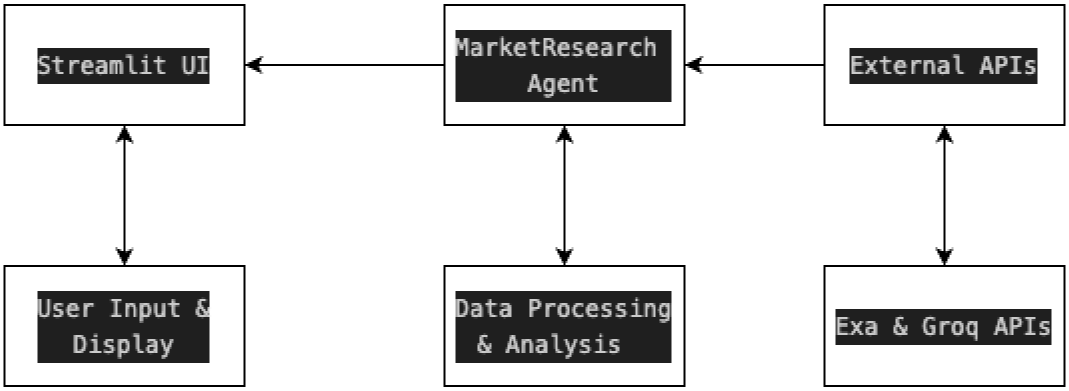
AI Strategy Analysis Tool - System Design Document

1. High-Level System Design

1.1 System Overview

The AI Strategy Analysis Tool is a web-based application that leverages AI to generate strategic recommendations for companies looking to implement AI/ML solutions. The system analyzes a company, determines its industry, conducts market research, generates tailored AI use cases, and creates a comprehensive strategy proposal.

1.2 Architecture Diagram



1.3 Core Components

1.3.1 Streamlit UI

- Provides the user interface for interacting with the application
- Handles user input (company name)
- Displays progress indicators and results
- Organizes output into tabs for better user experience

- Enables report downloading

1.3.2 MarketResearchAgent

- Core business logic component
- Orchestrates the analysis workflow
- Interfaces with external APIs
- Processes and transforms data
- Generates the final proposal

1.3.3 External API Integration

- **Exa API**: Provides web search capabilities for market research
- **Groq API**: Delivers AI-powered text generation for analysis and recommendations

1.4 Data Flow

1. **User Input**: User enters a company name in the Streamlit UI
2. **Industry Determination**: System identifies the company's industry using Groq API
3. **Market Research**: System conducts research using Exa API based on industry and company
4. **Use Case Generation**: System generates AI/ML use cases using Groq API
5. **Resource Collection**: System finds relevant datasets for each use case using Exa API
6. **Proposal Generation**: System creates a comprehensive proposal document
7. **Result Display**: System presents the results to the user in an organized format

1.5 Key Technologies

- **Frontend**: Streamlit (Python-based web application framework)

- **Backend**: Python (core logic and API integration)
- **AI/ML**: Groq API (LLM for text generation)
- **Data Retrieval**: Exa API (web search)
- **Environment Management**: Python-dotenv (configuration)

2. Low-Level System Design

2.1 Component Details

2.1.1 MarketResearchAgent Class

```
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)
```

2.1.2 Main Function

```
def main():
    # UI setup
    # Input handling
    # Analysis process orchestration
    # Result display
```

2.2 Method Specifications

2.2.1 `determine_industry(self, company_name)`

- **Purpose**: Identify the primary industry of a company
- **Input**: Company name (string)
- **Output**: Industry name (string)

- **Process**:
    1. Construct a prompt for industry classification
    2. Send prompt to Groq API using llama-3.1-8b-instant model
    3. Process response to extract industry name
    4. Apply fallback mechanisms if needed
- **Error Handling**: Returns "Technology" as fallback if API fails

2.2.2 `research_company(self, company_name)`

- **Purpose**: Gather market research on company and industry
- **Input**: Company name (string)
- **Output**:
    1. Research insights (dictionary)
    2. Industry (string)
- **Process**:
    1. Determine industry using `determine_industry` method
    2. Construct search queries for industry trends, company position, and innovations
    3. Execute searches using Exa API
    4. Format and structure results
- **Error Handling**: Displays error message if search fails

2.2.3 `generate_use_cases(self, company_name, industry, research_insights)`

- **Purpose**: Generate AI/ML use cases for the company

- **Input**:
    1. Company name (string)
    2. Industry (string)
    3. Research insights (dictionary)
- **Output**: List of use cases (list of strings)
- **Process**:
    1. Construct prompt with company and industry context
    2. Send prompt to Groq API
    3. Parse response into separate use cases
- **Error Handling**: Returns generic use case if API fails

2.2.4 `collect_resource_assets(self, use_cases)`

- **Purpose**: Find relevant datasets for each use case
- **Input**: Use cases (list of strings)
- **Output**: Mapping of use cases to resources (dictionary)
- **Process**:
    1. For each use case, search for datasets on platforms like Kaggle, Hugging Face, GitHub
    2. Filter results for relevance
    3. Create mapping between use cases and resources
- **Error Handling**: Returns empty list if search fails

2.2.5 `generate_final_proposal(self, company_name, industry, use_cases, resource_map, research_insights)`

- **Purpose**: Create comprehensive strategy proposal
- **Input**:
    1. Company name (string)
    2. Industry (string)
    3. Use cases (list)
    4. Resource map (dictionary)
    5. Research insights (dictionary)
- **Output**: Markdown-formatted proposal (string)
- **Process**:
    1. Create structured markdown document
    2. Add research insights section
    3. Add use cases section with resources
    4. Format for readability
- **Error Handling**: Returns error message in proposal format if generation fails

2.3 API Integration Details

2.3.1 Exa API Integration

- **Client Initialization**: `self.exa = Exa(api_key=os.getenv("EXA_API_KEY"))`
- **Search Method**: `self.exa.search(query, num_results=3, type="neural")`

Response Processing:

```
 research_results[query] = [
    {"title": res.title, "url": res.url, "snippet": res.summary}
    for res in results.results
]
```

-
- **Error Handling**: Try-except blocks around API calls

2.3.2 Groq API Integration

- **Client Initialization**: `self.groq_client = groq.Client(api_key=os.getenv("GROQ_API_KEY"))`

LLM Configuration:

```
 self.llm_config = {
    "config_list": [{
        "model": "llama-3.1-8b-instant",
        "api_key": os.getenv("GROQ_API_KEY"),
        "api_type": "groq"
    }]
}
```

-
- **Request Format**: Using autogen's AssistantAgent for message formatting
- **Error Handling**: Try-except blocks around API calls

2.4 Data Structures

2.4.1 Research Insights

```
{
    "query1": [
        {"title": "Article Title", "url": "URL", "snippet": "Summary Text"},
        # More results...
    ],
    "query2": [
        # Results for query2...
    ],
    # More queries...
}
```

2.4.2 Resource Map

```
{
    "Use Case 1": [
        {"title": "Dataset Title", "url": "Dataset URL"},
        # More datasets...
    ],
    "Use Case 2": [
        # Datasets for Use Case 2...
    ],
    # More use cases...
}
```

2.5 Error Handling Strategy

2.5.1 API Failure Handling

- All API calls are wrapped in try-except blocks
- Errors are displayed to the user via Streamlit's error display
- Default fallbacks are provided for critical functions

2.5.2 Input Validation

- Company name is required before analysis begins
- Progress indicators show the current state of the analysis

2.5.3 Response Processing

- Robust parsing of LLM responses with fallback mechanisms
- Pattern matching to extract relevant information from verbose responses

2.6 Performance Considerations

2.6.1 API Rate Limiting

- Exa API: Limited to a certain number of searches per month
- Groq API: Limited by tokens per minute (TPM)

2.6.2 Optimization Strategies

- Limit the number of search results to reduce API usage
- Use efficient prompts to minimize token usage
- Implement caching for repeated queries (future enhancement)

2.7 Security Considerations

2.7.1 API Key Management

- API keys stored in .env file (not committed to version control)
- Keys loaded using python-dotenv

2.7.2 Data Privacy

- No user data is stored permanently
- Reports are generated on-demand and can be downloaded by the user

3. Future Enhancements

3.1 Technical Improvements

- Implement caching to improve performance and reduce API calls
- Add support for more data sources
- Implement asynchronous processing for parallel API calls

3.2 Feature Enhancements

- Add support for comparing multiple companies
- Implement visualization of industry trends
- Add customization options for the analysis process

3.3 UI Improvements

- Add more interactive elements
- Implement real-time feedback during analysis
- Add support for different report formats (PDF, DOCX)

4. Appendix

4.1 Dependencies

- streamlit>=1.30.0
- autogen>=0.2.0
- python-dotenv>=1.0.0
- exa_py>=0.1.0
- groq>=0.4.0

4.2 Environment Variables

- GROQ_API_KEY: API key for Groq
- EXA_API_KEY: API key for Exa

4.3 References

- Streamlit Documentation
- Groq API Documentation
- Exa API Documentation