High-Level Design (HLD)

An overview of the system's major components, their responsibilities, and interactions.

Components and Data Flow

- 1. Frontend (User Interface)
 - Streamlit Interface: Handles user input and displays output in a web-based UI.
 - Inputs:
 - API Key, Company Name, Industry, and Focus Areas are collected from the user.
 - Display:
- Displays generated use cases in a structured format with titles, objectives, applications, and benefits.

2. Backend Logic

- ResearchAgent:
 - Accepts the API key and uses it to authenticate with the external search API.
 - Builds a guery based on the company name and industry.
 - Fetches industry-specific information about AI use cases.
- UseCaseAgent:
- Generates specific AI use cases based on the information returned by 'ResearchAgent'.
- Uses predefined templates to produce titles, objectives, applications, and benefits for each focus area.

3. External API:

- Search API:
- An external search API (such as 'serper.dev') is used to fetch information about AI applications within a specified industry.
 - Response Parsing:
- The application parses the JSON response from the API to extract relevant information for each use case.

High-Level Data Flow

- 1. User Inputs -> Streamlit Interface
 - User provides inputs like API Key, Company Name, and Industry.
- 2. Streamlit Interface -> ResearchAgent
 - Passes user inputs to 'ResearchAgent', which makes an API call.
- 3. ResearchAgent -> External API
 - Sends a search request to fetch industry-specific AI applications.
- 4. External API -> ResearchAgent
 - Returns JSON data containing relevant AI applications.
- 5. ResearchAgent -> UseCaseAgent
 - Provides the fetched industry information to 'UseCaseAgent'.
- 6. UseCaseAgent -> Streamlit Interface
 - Returns the generated use cases, which are then displayed to the user.

__.

Objective: Detail each module's internal logic, including classes, functions, and data structures.

1. Streamlit Interface

- Widgets:
- 'st.text_input': Collects 'API Key', 'Company Name', and 'Industry'.
- 'st.multiselect': Allows users to select multiple focus areas for use cases.
- 'st.button': Initiates use case generation upon click.
- Display Logic:
 - Loops through each use case generated by 'UseCaseAgent' and displays it in a structured layout.

2. 'ResearchAgent' Class

- Attributes:
 - 'api_key': Stores the API key needed for authentication.
- Methods:
 - 'search_company_info(company_name, industry)':
- Constructs a JSON payload containing the search query ('AI applications for {company_name} in {industry} industry').
- Sends a POST request to the external API and returns the JSON response or an error message if the request fails.

3. 'UseCaseAgent' Class

- Attributes:
 - 'focus_areas': A list of default focus areas, allowing dynamic customization.
- Methods:
 - 'generate_use_cases(company_info, company_name, focus_areas)':
 - Extracts industry information from 'company info'.
 - Iterates through each focus area to create use cases.
 - Uses a template-based approach to generate each use case with:
 - 'title': Title of the use case.
 - 'objective': Specific goal of the use case.
 - 'application': Description of the AI application for the focus area.
 - 'benefits': List of expected benefits from implementing the use case.

Data Structures

```
- Use Case Dictionary:
```

```
Each use case is represented as a dictionary:
""python
{
    "title": "Example Use Case Title",
    "objective": "Goal of the use case",
    "application": "Description of how AI is applied",
    "benefits": ["Benefit 1", "Benefit 2"]
}
```

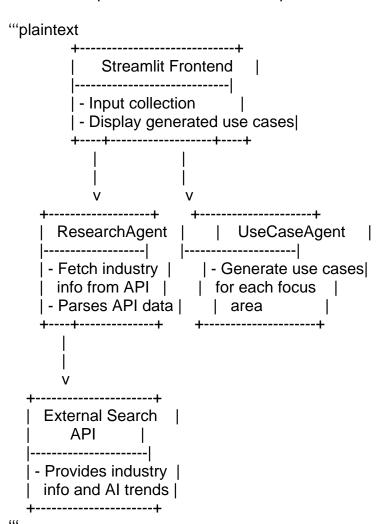
- JSON Response:
 - 'company_info': JSON response from the external API, parsed for relevant data.

Error Handling and Validation

- Error Handling:
 - If the API request fails, an error message is displayed using 'st.error'.
- Input Validation:
 - Ensures all required fields (API Key, Company Name, Industry) are filled before sending the request.

Architectural Diagram

Here's a simple architectural flow to represent the design visually:



.....