StackGen

StackGen - AI Dev Take-Home Assessment

Overview

Multi-Agent System Design

Design and implement a multi-agent system that intelligently routes queries to specialized agents with distinct capabilities and integrations.

Time Estimate: 4-6 hours
Language: Python (preferred) or your choice
Submission: GitHub/Gitlab/Bitbucket repository with working code + 2 minute video

Problem Statement

Build a multi-agent orchestration system with the following architecture:

Agent Hierarchy

Agent A: GitHub Agent

- Integration A1: **GitHub User 1** (Personal Access Token for User 1)
    - Access: User 1's repos, issues, PRs, stars
- Integration A2: **GitHub User 2** (Personal Access Token for User 2)
    - Access: User 2's repos, issues, PRs, stars

Agent B: Linear Agent

- Integration B1: **Linear User 1** (API key for User 1)
    - Access: User 1's issues, projects, teams
- Integration B2: **Linear User 2** (API key for User 2)
    - Access: User 2's issues, projects, teams

System Requirements

Your system must:

1. **Intelligent Routing**: Analyze user queries and route to the appropriate agent (GitHub or Linear)

StackGen

2. **Integration Selection**: Each agent should automatically select the correct user integration (User 1 or User 2) based on the query

3. **Clarification Handling**: When the user identity is ambiguous, ask for clarification
    - Example: "Show me open pull requests" → should clarify which user (User 1 or User 2)
    - Example: "Show me Alice's pull requests" → if Alice is User 1, use Integration A1
4. **Out-of-Scope Handling**: For queries unrelated to GitHub or Linear, respond with: *"I cannot answer this question"*

5. **Response Quality**: Provide meaningful responses using the selected integration with real user data

Example Interactions

Assume User 1 = "Alice" and User 2 = "Bob" for these examples:

```
None
User: "Show me Alice's open pull requests"
System: [Routes to Agent A (GitHub) → Integration A1 (User 1)]
Response: "Alice has 3 open pull requests:
1. Fix auth bug in user-service (#42)
2. Add unit tests for API endpoints (#38)
..."

User: "What issues are assigned to Bob in Linear?"
System: [Routes to Agent B (Linear) → Integration B2 (User 2)]
Response: "Bob has 5 issues assigned:
- [ENG-123] Implement user authentication (In Progress)
- [ENG-145] Fix memory leak in worker (Todo)
..."

User: "List Alice's repositories"
System: [Routes to Agent A (GitHub) → Integration A1 (User 1)]
Response: "Alice has 12 repositories:
1. ml-pipeline
2. api-gateway
..."

User: "Show me Bob's Linear issues that are in progress"
System: [Routes to Agent B (Linear) → Integration B2 (User 2)]
```

StackGen

Response: "Bob has 2 issues in progress:
- [ENG-189] Refactor authentication module
- [ENG-201] Update API documentation
..."

User: "Show me open pull requests"
System: [Routes to Agent A (GitHub) → Clarification needed]
Response: "I can help with that! Which user's pull requests would you like to see - Alice's or Bob's?"

User: "What issues are high priority?"
System: [Routes to Agent B (Linear) → Clarification needed]
Response: "I can help with that! Which user's issues would you like to see - Alice's or Bob's?"

User: "What's the weather today?"
System: [Out of scope]
Response: "I cannot answer this question"

API Setup Guide (for candidates)

You'll need to set up credentials for 2 different users for both GitHub and Linear. Feel free to create dummy accounts. Linear can be a single org with both users as members:

GitHub API - User 1

```shell
export GITHUB_TOKEN_USER1="ghp_user1_token_here"
export GITHUB_USERNAME_USER1="username-1"
```

GitHub API - User 2

```shell
export GITHUB_TOKEN_USER2="ghp_user2_token_here"
export GITHUB_USERNAME_USER2="username-2"
```

StackGen

Linear API - User 1

```shell
export LINEAR_API_KEY_USER1="lin_api_user1_key_here"
export LINEAR_USERNAME_USER1="username-1"
```

Linear API - User 2

```shell
export LINEAR_API_KEY_USER2="lin_api_user2_key_here"
export LINEAR_USERNAME_USER2="username-2"
```

Implementation Guidelines

You have complete freedom in your approach. Here are some options:

- **Language**: Python preferred, but use whatever you're comfortable with
- **Agent Framework**: You can use:
  - LangChain, CrewAI, AutoGen, or similar frameworks
  - Feel free to use Model Context Protocol (MCP) or Agent-to-Agent (A2A) tools as required
  - Your own custom implementation from scratch
  - Any combination of the above
- **LLM**: Use any LLM you prefer:
  - OpenAI (GPT-4, GPT-3.5)
  - Anthropic (Claude)
  - Open-source models (Llama, Mistral, etc.)
  - Or even rule-based logic without an LLM if you prefer!
- **Architecture**: Design it however makes sense to you - we want to see your thinking
- We **RECOMMEND** leveraging AI coding tools to help out with coding, pair-programming, debugging, architecture-design etc.

What we care about:

- Does it work as specified?
- Is the code readable and organized?
- Can we run it by following your README?

StackGen

- Did you handle the core requirements (routing, user selection, clarification, out-of-scope)?

What we don't care about:

- Perfect code - we know this is a time-boxed exercise
- Production-ready error handling for every edge case
- Extensive test coverage (though tests are a nice bonus!)
- Performance optimization

Bonus Points (Optional - Pick What Interests You!)

These are completely optional. Choose ones that showcase your strengths:

- **Docker**: Containerize your application
- **Documentation**: Architecture diagram or detailed design explanation
- **Testing**: A few unit tests for key logic
- **Logging**: Show which agent/user was selected and why
- **Extensibility**: Make it easy to add User 3, User 4, etc.
- **Caching**: Cache API responses to avoid rate limits
- **Observability**: Log the decision-making process

Pick 1-2 bonus items that interest you rather than trying to do them all!

Submission Requirements

Minimum Requirements

Your code repository must include:

1. **Source Code**: Your implementation
2. **.env.example**: Show what environment variables are needed (no real tokens!)
3. **README.md** with:
    - A paragraph or two on what you built and how it works
    - How to set it up and run it
        i. Ensure you include your requirements.txt/package.json etc.
    - Any assumptions or limitations

Questions?

If you have clarifying questions about the requirements, please document your assumptions in the README. We're interested in seeing how you handle ambiguity and make pragmatic decisions.

StackGen

Good luck! We're excited to see your solution. Remember: working code with clear documentation beats perfect code every time.