## Multi-Agent System for AI TechHub Embassy Staff

### Sequence Diagram: Initial Use-Case Intake & Agent Delegation (Text Version)

**Actors:**

* User (Sales, Builder, etc.)
* ConciergeAgent
* OrchestratorAgent
* NavigatorAgent
* ArchivistAgent

**Flow:**

1. User logs in, visits TechHub Resource site, and initiates an ‘embassy-chat’ session.
2. ConciergeAgent greets user by name and starts the intake dialogue (e.g., “Is this a NEW project or an EXISTING (pick one) project we will discuss today?”)
3. Following sequence assumes a NEW project is chosen:
4. ConciergeAgent pops an Intake Form, explaining that they are standing by to help the User fill it out with as few or as many known or guessed details as currently available. Example fields:
   1. Business or technical use-case description
   2. Industry vertical
   3. Specific client and their use-case context
   4. Internal NTT contacts involved
   5. Client cloud choice
   6. Scope-limiting project constraints like time and budget
   7. Expected delivery timeline or deadlines
   8. Primary success criteria / goals
   9. Known dependencies or blockers
   10. Desired TechHub resource types (Demo, Solution, Component)
   11. Stage of engagement (Discovery, Design, Build, Pilot, etc.)
   12. Known compliance or security requirements …to complete as much of the initial Use-Case JSON structure as currently possible.
5. ConciergeAgent passes structured intake to the OrchestratorAgent
6. OrchestratorAgent analyzes intent, spawning the appropriate Agents-On-Demand (in this case, a NavigatorAgent)
7. NavigatorAgent queries TechHub resource catalog (Demos, Solutions, Components, Registered/Discoverable Internal Agents)
8. NavigatorAgent generates a draft BOM and resource suggestions
9. NavigatorAgent reports findings back to OrchestratorAgent
10. OrchestratorAgent relays result to ConciergeAgent
11. ConciergeAgent presents results to User
12. ArchivistAgent logs interaction and project state (timestamped)

**Mermaid Markup & Resulting Diagram:**

title Tech-Hub User Initiates NEW Project, Receives BOM + Resource Suggestions

participant User

participant ConciergeAgent

participant OrchestratorAgent

participant NavigatorAgent

participant ArchivistAgent

User->>ConciergeAgent: Initiates embassy-chat session

ConciergeAgent->>User: Greets user by name\n(asks NEW or EXISTING project)

User->>ConciergeAgent: Selects NEW project

ConciergeAgent->>User: Presents Intake Form\n(KISS format with guidance)

ConciergeAgent->>OrchestratorAgent: Sends structured intake JSON

OrchestratorAgent->>NavigatorAgent: Spawns agent to process intake

NavigatorAgent->>NavigatorAgent: Query resource catalog\n(Demos, Solutions, Components, Agents)

NavigatorAgent->>OrchestratorAgent: Return BOM + resource suggestions

OrchestratorAgent->>ConciergeAgent: Relay results

ConciergeAgent->>User: Presents matched resources

ConciergeAgent->>ArchivistAgent: Log project state + interaction

**A diagram with text and words

AI-generated content may be incorrect.**

### Component Diagram: Logical View of Embassy Agent System (Text Version)

**User Interface:**

* Embedded Web Chat (via iframe or React component)

**Core Agents (Dynamic):**

* OrchestratorAgent: Routes tasks, spawns and manages agents
* ConciergeAgent: Handles dialogue, intake, user guidance
* NavigatorAgent: Searches resource catalog, suggests components
* ArchivistAgent: Stores dialogue history, project status, user metadata

**Pluggable Future Agents:**

* ResearchAgent: Searches precedent patterns, client analogs
* InfraAgent: Validates or requests infra needs
* ComplianceAgent, CostAgent, etc.

**Knowledge & Storage:**

* TechHub Resource Catalog (API/RAG/Vector Search)
* TechHub Project Tracker (DB or JSON storage)
* Persistent User Session Storage (Chat History, Project Linkage)

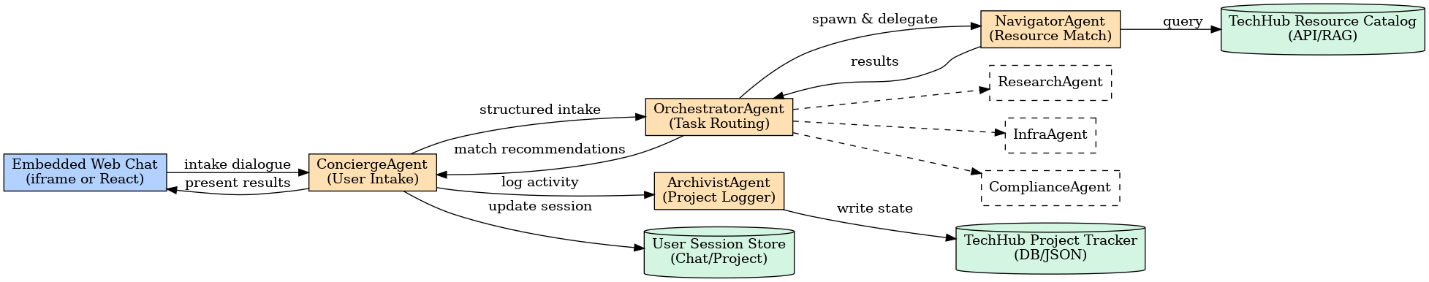
**Infrastructure Context:**

* Azure Functions / Container Apps
* Azure OpenAI + Azure Cognitive Search
* CosmosDB / Azure Blob Storage for state and memory

**External Integration (Optional):**

* Power Platform for dashboard views
* GitHub or Azure DevOps for accepted POCs

**Component Diagram Visualization**



### Notes:

* This diagram assumes dynamic agent orchestration from a central orchestrator (inspired by Swarm patterns).
* All agents are stateless but operate with context pulled from a central session/project state blob.
* MVP includes ConciergeAgent, OrchestratorAgent, NavigatorAgent, and ArchivistAgent.
* Chat session persists via embedded frontend and links back to stored project ID.

### Unified Schema Summary

#### UseCase

{  
 "use\_case\_id": "string (UUID)",  
 "title": "string",  
 "description": "string",  
 "industry\_vertical": "string",  
 "client\_name": "string",  
 "client\_context": "string",  
 "internal\_contacts": ["string"],  
 "cloud\_preference": "string",  
 "project\_constraints": {  
 "budget": "string or number",  
 "timeline": "string or date",  
 "known\_dependencies": ["string"],  
 "compliance\_requirements": ["string"]  
 },  
 "engagement\_stage": "string",  
 "success\_criteria": ["string"],  
 "resource\_type\_preference": ["Demo", "Solution", "Component"],  
 "status": "string",  
 "created\_by": "string",  
 "created\_at": "ISO timestamp",  
 "last\_updated": "ISO timestamp"  
}

#### TechHubProject

{  
 "project\_id": "string (UUID)",  
 "use\_case\_id": "string",  
 "title": "string",  
 "current\_phase": "string",  
 "created\_by": "string",  
 "collaborators": ["string"],  
 "agent\_activity\_log": [  
 {  
 "agent": "string",  
 "timestamp": "ISO timestamp",  
 "action": "string",  
 "summary": "string"  
 }  
 ],  
 "status\_notes": "string",  
 "last\_updated": "ISO timestamp",  
 "archived": "boolean",  
 "promoted\_to\_resource\_catalog": "boolean",  
 "final\_linked\_assets": ["string"],  
 "repository\_url": "string",  
 "repository\_visibility": "string (Public, Private, Internal)"  
}

#### ResourceMatch

{  
 "match\_id": "string (UUID)",  
 "use\_case\_id": "string",  
 "matched\_on": "ISO timestamp",  
 "matched\_by": "string",  
 "recommended\_resources": [  
 {  
 "resource\_id": "string",  
 "title": "string",  
 "type": "string",  
 "relevance\_score": "number",  
 "description": "string",  
 "link": "string"  
 }  
 ],  
 "generated\_bom": [  
 {  
 "item": "string",  
 "category": "string",  
 "source": "string",  
 "required": "boolean"  
 }  
 ],  
 "notes": "string",  
 "status": "string"  
}

### PROMPT for Claude to use with Plan-Spec-Code Mode:

You are a senior systems engineer and Python architect. You’ve been asked to help prototype an intelligent multi-agent system called the “AI Embassy Staff” for the NTT DATA TechHub — a digital resource center full of Demos, Solutions, and Components.

---

🧭 Purpose:

To convert static TechHub resources into an interactive, agent-guided system that assists internal users (e.g., sales teams, builders, infra leads) in:

- Capturing and refining use-cases

- Matching those use-cases to existing TechHub resources

- Managing a project lifecycle from intake to promotion into the catalog

---

🛠️ What to Build:

Create a bootstrappable Python-based prototype using modular agents coordinated via an orchestrator pattern. The prototype should:

- Handle chat intake (CLI or web stub)

- Collect structured data and initialize a `UseCase` and `TechHubProject`

- Spawn agents that query a simulated resource catalog and return a `ResourceMatch`

- Persist all data as JSON, ideally using CosmosDB-style mock or flat file

Use LangGraph or CrewAI if multi-agent orchestration is required. FastAPI can be used if designing an HTTP interface layer. Assume deployment will eventually be in Azure Functions or Container Apps, with storage via CosmosDB or Blob Store.

---

📐 System Components:

- \*\*OrchestratorAgent\*\* — reads user goals, spawns appropriate agents

- \*\*ConciergeAgent\*\* — manages chat flow and guides intake

- \*\*NavigatorAgent\*\* — matches use-case to TechHub resources and drafts a BOM

- \*\*ArchivistAgent\*\* — stores session metadata and project state

All agents are stateless and interact through project/session memory.

---

📁 Data Models (Use these exactly):

### UseCase

[see UseCase in Unified Schema Summary]

### TechHubProject

[see TechHubProject in Unified Schema Summary]

### ResourceMatch

[see ResourceMatch in Unified Schema Summary]

---

📂 Output File Structure:

- `/agents/`: Python files for each agent class

- `/models/`: Typed Python classes (Pydantic or dataclass) for schemas above

- `main.py`: Entry point to initiate the orchestrator and a demo interaction

- `api.py`: (Optional) FastAPI stub for chat interface

- `.env` or `/config/`: Configs for local simulation or dev keys

- `README.md`: Setup, run instructions, expansion notes

---

🎯 Deliverables:

- Code that simulates the project lifecycle from use-case intake to resource matching

- Agent flow and data logging, even if mock-based

- Designed for future expansion with additional agents (InfraAgent, ComplianceAgent, etc.)

Focus on core routing and lifecycle capture — front-end and UI polish can come later.