Mindhive Assessment

This technical assessment is for Mindhive's AI SoftwareEngineer role. You will have **7 days** to complete all parts, demonstrating your ability to design multi-turn conversations, implement agentic planning, integrate external tools and custom APIs, handle unhappy flows, and build RAG pipelines.

Expected key points include:

- **State Management & Memory** (tracking slots/variables across turns)
- **Planner/Controller Logic** (intent parsing, action selection, follow-up questions)
- **Chat Interface** (front end UI to interact with the chatbot)
- **Tool Integration** (calling a calculator API, error handling)
- **Custom API Consumption & Testing** (FastAPI endpoints for restaurants, product KB, outlets)
- **Retrieval-Augmented Generation** (vector-store plus Text2SQL for domain data)
- **Robustness** (graceful degradation on missing input, downtime, or malicious payloads)

Part 1: Sequential Conversation

Objective: Keep track of at least three related turns.

- **Code-First Option:** Use LangChain or any Python-based framework to implement memory or state.

Example Flow:

1. User: "Is there an outlet in Petaling Jaya?"
2. Bot: "Yes! Which outlet are you referring to?"
3. User: "SS 2, whats the opening time?"
4. Bot: "Ah yes, the SS 2 outlet opens at 9:00AM"

Deliverables:

- Exported flow or framework project
- Automated tests for both happy and interrupted paths

Part 2: Agentic Planning

Objective: Show how the bot decides its next action (ask, call a tool, or finish).

Implement a simple planner/controller loop in your chosen environment that:

1. Parses intent and missing information
2. Choose an action (e.g., ask a follow-up, invoke calculator, call RAG/Text2SQL endpoint)
3. Executes that action and returns the result

Deliverables:

- Planner/controller code
- A short write-up of decision points

Part 3: Tool Calling

Objective: Integrate a Calculator Tool for simple arithmetic.

Agents must detect arithmetic intent, invoke the calculator endpoint, parse responses, and handle errors without crashing.

Deliverables:

- Calculator API integration code or definition
- Example transcripts showing both successful calculations and graceful failure handling

Part 4: Custom API & RAG Integration

Objective: Build and consume FastAPI endpoints for domain data and test them as your external APIs.

1. **Product-KB Retrieval Endpoint**
    - Ingest ZUS product docs into a vector store (e.g., FAISS, Pinecone).
        i. Source: https://shop.zuscoffee.com/ > Drinkware only
    - Expose `/products?query=<user_question>` that retrieves top-k and returns an AI-generated summary.
2. **Outlets Text2SQL Endpoint**
    - Maintain a SQL DB of ZUS outlets (location, hours, services).
        i. Source: https://zuscoffee.com/category/store/kuala-lumpur-selangor/
    - Expose `/outlets?query=<nl_query>` that translates to SQL, executes it, and returns results.

Deliverables:

- FastAPI repo with OpenAPI spec covering `/products`, `/outlets`
- Vector-store ingestion scripts and retrieval code for products
- Text2SQL prompts/pipeline plus DB schema and executor for outlets
- Chatbot integration code demonstrating calls to all three endpoints
- Example transcripts for each endpoint showing success and failure modes

Part 5: Unhappy Flows

Objective: Ensure robustness against invalid or malicious inputs.

Test at least three cases across your integrations:

- Missing parameters (e.g., user says “Calculate” or “Show outlets” without args)
- API downtime (simulate HTTP 500 on any endpoint)
- Malicious payload (e.g., SQL injection attempt in `/outlets`)

Bot should respond with clear error messages, recovery prompts, and never crash.

Deliverables:

- Test suite covering these negative scenarios
- Summary of your error-handling and security strategy

Part 6: Frontend Chat UI (No Streamlit/Gradio)

Objective: Ship a chat interface that exercises Parts 1–5 end-to-end. It must visualize agentic planning, surface tool/RAG activity, handle unhappy flows gracefully, and be easy to run locally.

Restrictions: Do not use Streamlit, Gradio, or similar “app builders.” You may use React or Vue (preferred) or plain HTML + JS.

Functional Requirements

- Chat Surface
- Message list with avatars, timestamps, and multi-turn threading.
- Composer with: multiline input, enter-to-send, and “shift+enter” for newline.
- Quick actions: /calc, /products, /outlets, /reset (autocomplete as you type).
- Persist conversation to localStorage so a refresh doesn’t lose state.
- Update live after each turn (reflects Part 1 memory behavior).

Submission Checklist

- **GitHub Repo:** Public link (no secrets)
- **Hosted Demo:** e.g., Heroku, Vercel URL
- **README.md:**
    - Setup & run instructions
    - Architecture overview and key trade-offs
- **Documentation:**
    - API specification (including RAG & Text2SQL endpoints)
    - Flow diagrams or screenshots of your chatbot setup

Delivery: Send your GitHub link and demo URL to jermaine@mindhive.asia (cc: johnson@mindhive.asia, ivan@mindhive.asia)