Task 1 – Design Explanation

Objective

Design a reliable LLM-based system that converts unstructured business text into **strictly valid, schema-compliant JSON**, while minimizing hallucinations and handling malformed model output gracefully.

The system must: - Produce consistent structured output - Never crash on invalid or partial LLM responses - Explicitly handle uncertainty and missing data

---

Overall Design Philosophy

The system follows a **zero-trust LLM architecture**:

> The LLM is treated as a semantic extractor, not a source of truth.
> All correctness guarantees are enforced in deterministic Python code.

This is achieved using a **constraint-first prompt**, **safe JSON parsing**, and a **post-generation validation layer**.

---

1. Input Handling & Auto-Detection

The system first analyzes the input text to determine whether it represents:

- A **single material request**
- A **batch of multiple requests** (multiline or pasted blocks)

Design Choice

- Single request → LLM is instructed to return **one JSON object**
- Multiple requests → LLM is instructed to return a **JSON array**, with one element per input line

This prevents invalid outputs such as multiple top-level JSON objects, which are not parseable by standard JSON parsers.

---

2. Prompt Design (Constraint-First)

Each LLM call uses a tightly constrained prompt that includes:

- The **exact JSON schema** (field names, types, allowed values)

- Explicit instructions:
  - "Return ONLY valid JSON"
  - "No markdown, no explanations, no extra text"
- Explicit null-handling rules:
  - Missing or uncertain information must be returned as `null`
- Clear semantic rules:
  - Urgency inference logic
  - Date parsing expectations

Temperature is set to **0.1** to reduce randomness and improve consistency.

---

3. Hallucination Control Strategy

Hallucinations are controlled using **two layers**:

Prompt-Level Controls

- Explicit "do not invent or hallucinate" instructions
- Domain-specific grounding rules for urgency and deadlines
- Clear examples of when to return `null`

Code-Level Controls

- The system never trusts the raw LLM output
- A validation layer:
  - Drops extra or unexpected fields
  - Normalizes null values
  - Enforces allowed urgency values
  - Converts invalid or ambiguous dates to `null`

---

4. Robust JSON Parsing (Critical Design Decision)

Single-Parse Rule

The system parses JSON **exactly once** using: