Software Engineering Take-Home Case

Human-in-the-Loop Order Management System

Introduction

Hi there!

Great news - if you're reading this it means we think you have what it takes to join the Momentum Digital team. Of course, we can't just trust our gut, so we've created this challenge to see how you work.

We set up this challenge to see strength in the following areas:

1. Writing clean, well-structured code and making smart architectural choices
2. Picking up and integrating new technologies quickly (like LLMs or external APIs)
3. Staying pragmatic when building under time pressure
4. Creating software that's usable and maintainable
5. Clearly explaining your reasoning, trade-offs, and decision-making process

Mission

As a new engineer at Momentum Digital, you've been handed a dataset from a client who gets order requests via email. Right now, they're manually processed, but we're building Human-in-the-loop systems where AI extracts the data, humans review it, and over time, LLMs handle more automatically. This mirrors our real IGEPA Order Agent project.

The task

Build a Django application that demonstrates a human-in-the-loop order processing flow: provide a review interface for extracted orders, integrate an LLM to (re)extract or validate order data, and post approved orders to an external API. Deliver something actionable and reliable within a **6-hour** timeframe; use AI assistants, Stack Overflow, or any docs as needed—this evaluates how you work and think, not memorization.

Scope:

1. Provide a review interface to view, edit, approve, or reject orders
2. Integrate an LLM API to implement or improve extraction/validation of order information
3. Post approved orders to an external API (the endpoint may be mocked or handled gracefully if unavailable)

As with most engineering projects, there are multiple angles (see Main Technical Challenges below). Tackle at least one in-depth, while briefly addressing the others (e.g., with a proposed roadmap). Play to your strengths—impress us with what you do best!

Requirements

The human-in-the-loop order review application must meet the following requirements:

1. Functional review interface

Provide functionality to review and manage orders extracted from emails:

Implement a flow that allows:

- View incoming orders that need review
- Edit order details if the extraction isn't perfect
- Approve orders (which triggers sending them to the external API)
- Reject orders with notes explaining why

Structure is up to the implementer. Demonstrate an approach to:

- Designing data models
- Building a usable interface
- Managing order state throughout the workflow

2. Data Validation

Ensure all order data is valid before it can be approved:

- Customer information must be complete
- Products must have quantities and prices
- Totals should be calculated correctly
- Handle missing or malformed data gracefully

Don't let invalid data slip through. Show us how you ensure data integrity.

3. LLM API Integration

Implement functionality to (re)extract or validate order information using an LLM API:

- Could be a button to "re-parse" an order
- Could be automatic enhancement for low-confidence extractions
- Could be validation of extracted data

Whatever approach you choose, demonstrate:

- Proper API communication

- JSON request/response handling
- Error recovery (timeouts, rate limits, invalid responses)
- How you update the order data with LLM results

You can use any LLM provider (OpenAI, Anthropic, etc.) or mock the API call.

4. External API Communication

When an order is approved, POST it to the external order processing system.

API Endpoint: `POST http://localhost:8001/api/submit-order`

Request Format:

```
{
  "orderRequest": {
    "customer_name": "ACME Corporation",
    "reference": "ORDER-2024-001",
    "delivery_date": "2024-02-15",
    "comment": "Please deliver before 2 PM",
    "products": [
      {
        "sku": "PROD-123",
        "product_name": "A4 Paper Premium",
        "quantity": 50,
        "unit_price": 12.99
      },
      {
        "sku": "PROD-456",
        "product_name": "Toner Cartridge XL",
        "quantity": 3,
        "unit_price": 89.50
      }
    ],
    "total_amount": 918.00
  }
}
```

Expected Success Response (200 OK):

```json
{
  "success": true,
  "order_id": "EXT-ORDER-98765",
  "message": "Order successfully created"
}
```

Error Response (4xx/5xx):

```json
{
  "success": false,
  "error": "Invalid product SKU: PROD-123",
  "code": "VALIDATION_ERROR"
}
```

Note: This endpoint doesn't actually exist. Handle this appropriately—whether you mock it, build a dummy endpoint, or just demonstrate proper error handling.

5. Authentication

Implement basic user authentication so the review interface isn't publicly accessible.

Main Technical Challenges

As with most software projects, there are multiple dimensions to this challenge. Candidates should tackle these with varying depth—focusing on strengths and on what is most important for the 6-hour timeframe. Perfection isn't required; prioritization and approach matter.

1. System Architecture & Design

How do you structure a human-in-the-loop system that balances automation with human oversight? Consider:

- Data modeling: How do you represent emails, orders, and their lifecycle?
- State management: How do orders flow through the system?
- API architecture: How do you organize external integrations (LLM, order submission)?
- Separation of concerns: How do you keep business logic, persistence, and presentation separate?
- What design patterns or architectural principles guide your decisions?

2. Integration & External Dependencies

You need to work with LLMs and external APIs - things you don't control. How do you design around this? Consider:

- How do you structure the LLM integration so it's maintainable and testable?
- What happens when APIs are slow, unavailable, or return unexpected data?
- How do you prevent your system from being tightly coupled to specific providers?
- How would you swap out the LLM provider or the external order API?
- What's your strategy for handling API costs and rate limits?

3. Usability & Maintainability

Software isn't useful if people can't operate it or if it's a nightmare to maintain. Consider:

- User experience: How intuitive is your interface for reviewing orders?
- Code quality: If someone needs to modify this in 6 months, can they?
- Deployment: How would someone actually run this locally?
- Debugging: If this breaks, how would you figure out what went wrong?
- Testing: How do you verify it works correctly?

Note: Perfect execution is not required. Demonstrate reasoning and approach. Where trade-offs are made or scope is reduced, document why. **Play to strengths**—backend-focused entries can emphasize API design and data modeling; UX-focused entries can highlight thoughtful interfaces. Perfection is not expected.

The Data

The materials folder contains:

1. Database Files (JSON format)

The dataset includes six JSON files representing the relational database:

`data_generation/emails.json` - 32 emails (26 orders, 6 non-orders)
`data_generation/orders.json` - 26 processed orders
`data_generation/order_lines.json` - ~60 product line items
`data_generation/contacts.json` - ~16 contacts (people who send emails)
`data_generation/companies.json` - ~14 companies
`data_generation/order_emails.json` - Junction table linking emails to orders

The data includes:

- Clean, complete orders with all fields populated
- Orders with missing information (no references, missing delivery dates)
- Non-order emails (inquiries, spam, support requests)
- Multiple emails from the same contact (tests relationship handling)
- One company with multiple contacts (tests data modeling)
- Products with similar names (tests validation/matching logic)

Some extra notes:

- All order data is in the `orders` and `order_lines` tables, not nested in emails
- Some fields will be `null` or missing - that's intentional
- Email bodies contain the original order text in both plain text and HTML
- All data has been anonymized but maintains realistic structure

If you're unsure about something - feel free to make assumptions, but, as always, mention this in your documentation.

2. Workflow Screenshots (`screenshots/`)

Screenshots showing our current order review flow at a client. Use these to understand the kind of interface users expect. This is inspiration, not a strict requirement—build it however makes sense to you.

3. API Specification

The external API contract (included in the requirements section above).

Deliverables

1. Working Code

- Complete Django app in public GitHub repo
- All dependencies specified
- Database setup included
- Clear setup instructions

2. README.md

Include these sections:

The README should include:

Setup Instructions

- Step-by-step commands to get the application running
- Any environment variables or configuration needed
- Assume familiarity with Django; make testing easy

Architecture Overview

- How the application is structured
- Key design decisions
- Database schema (brief diagram or explanation)
- Rationale for the chosen organization

Trade-offs & Decisions

- Rationale for chosen libraries or approaches
- Focus areas and deprioritized items
- Any deliberate shortcuts (and rationale)
- Known limitations or incomplete areas and the rationale

What You'd Do With More Time

If you had another week or month, what would you improve or add? Be specific:

- What features would you prioritize?
- What technical debt would you address?
- What would you refactor?
- Roughly how much time would each improvement take?

AI Development & Improvement

This is an AI-powered system. How would you make it better over time?

- How would you improve the LLM prompts and extraction accuracy?
- What data would you collect to train or fine-tune models?
- How would you measure if the AI is getting better?
- What's the path to reducing human intervention over time?

Deployment & Going Live

For production readiness, outline what would need to happen:

- Deployment approach and intended infrastructure
- Required monitoring, logging, and observability
- Scaling strategy for processing 1000+ orders/day
- Security considerations
- Gaps remaining before going live

Important: These sections don't need to be long - a few paragraphs each is fine. We're not judging writing quality, just whether you understand the broader picture beyond just writing code.

3. Demo

- 4-6 screenshots showing the application working, OR
- Short screen recording (max 2 minutes) demonstrating the flow

Evaluation Criteria

Area	What We Look For
Does it work?	Can we run it? Does it do what it claims?
Code structure	Organized, understandable, good separation of concerns
Data handling	Proper validation, storage, integrity checks
API integration	Clean patterns, error handling
Decision making	Smart trade-offs, pragmatic choices, documented assumptions

We value working code over perfect code. Show us how you think and build.

Technical Requirements

- **Time:** 6 hours (verified via commit history)
- **Required:** Python 3.9+, Django 4.x/5.x
- **Database:** Your choice
- **Frontend:** Django templates, HTMX, light JS - whatever works
- **Environment:** Development only

Remember: We're evaluating how you think and build, not checking boxes. If you decide to spend more time on data modeling and less on UI polish, explain why that's the right call. If your LLM implementation is basic but your error handling is rock solid, tell us why you prioritized that.

Submission

Submit your work as a **public GitHub repository**.

Send the repository link to kim@momentumdigital.com or kilian@momentumdigital.com.

Include all code, migrations, documentation. We'll review commit history.

Deadline: [To be confirmed when assigned]

Questions?

If something's unclear during the case, email us at kim@momentumdigital.com or kilian@momentumdigital.com. We'll respond within a few hours during business hours (9 AM - 6 PM CET).

Document reasonable assumptions in your README.

Final Notes

No single "right" solution exists. We evaluate:

- Problem-solving approach
- Code structure
- Trade-offs under time pressure
- Communication of decisions

Focus on delivering working, explainable code.

Good luck - we're excited to see what you build!