Mid-Level Engineers Persistent Candidates - Payments Take-Home Problems

This exercise evaluates how you think, structure, and solve problems, not just your ability to code a perfect solution. We're looking for clean, readable code, reasonable architecture, and thoughtful tradeoffs. We're not looking for perfection — we want to understand how you work and think.

Tools Required

You are expected to complete this exercise using Cursor, an AI-assisted IDE designed for developers.

- You can start with a free trial of Cursor if you don't already have an account.
- The goal is not just to write code, but to show how you collaborate with an AI development tool.

What We're Evaluating

- Clarity and maintainability of your code
- Approach to problem solving and handling edge cases
- Use of modern development practices (e.g., modularity, tests, API design)
- Your ability to work independently, yet communicate decisions clearly
- How effectively you use AI (via Cursor) to aid development
- Creativity in applying stretch goals or AI features (for senior-level roles)

Submission Guidelines

- Share a public GitHub repo or zipped file
- Include a `README.md` with:
  - Setup instructions
  - Description of fraud logic and LLM usage
  - Explanation of any assumptions or tradeoffs

How We Will Review Your Submission

We will:

- Run your project locally (or using Docker, if provided)
- Review your codebase structure and naming conventions
- Review your Cursor prompts history
- Discuss your approach in a follow-up interview
- Ask how you used AI to help you think through problems, debug, or generate code

---

Mid-Level Problem 1: Mini Payment Gateway Proxy with LLM Risk Summary

Design a lightweight backend API that simulates routing payment requests to Stripe or PayPal based on a simple fraud risk score. Use an LLM to generate a human-readable explanation of each decision.

Requirements

1. API Endpoint:

```
POST /charge
```

Request Payload:

```
{
  "amount": 1000,
  "currency": "USD",
  "source": "tok_test",
  "email": "donor@example.com"
}
```

Behavior:

- Internally simulate a fraud risk score (float between 0 and 1)
  - Use simple heuristics: large amount, suspicious domain (e.g., `.ru`, `test.com`)
- Based on the score:
  - Route to payment processor if score < 0.5
  - Block submission if score ≥ 0.5
- Generate a natural-language summary of the risk using an LLM

Example Response

```
{
  "transactionId": "txn_abc123",
  "provider": "paypal",
  "status": "success",
  "riskScore": 0.32,
  "explanation": "This payment was routed to PayPal due to a moderately high low score based on a large amount and a suspicious email domain."
}
```

- Log all transactions in memory with timestamp and metadata
- Handle input validation and basic error responses
- Include basic unit tests
- Use modern TypeScript conventions and structure

Stretch Goals

- Add a `GET /transactions` endpoint
- Cache frequent LLM prompts for performance
- Dockerize the app
- Add a basic fraud rule config file to make heuristics configurable

---

Mid-Level Problem 2: Subscription Billing Simulator

Build a backend service that simulates recurring donations for nonprofit supporters. You'll handle subscription creation, automatic charging based on schedule, and use an LLM to tag and summarize each campaign description.

Requirements

1. API Endpoints

- `POST /subscriptions`

```
{
  "donorId": "abc123",
  "amount": 1500,
```

```
  "currency": "USD",
  "interval": "monthly",
  "campaignDescription": "Emergency food and clean water for earthquake victims in Nepal"
}
```

    - Store each active subscription
    - Call an LLM to:
        - Tag the campaign (e.g., `"disaster relief"`, `"Nepal"`)
        - Summarize it in one sentence
        - Store these as part of the subscription record
- `DELETE /subscriptions/:donorId`
    - Cancels a recurring donation

2. Daily/Batched Processing

- Simulate a background job (e.g., `setInterval`) that:
    - “Charges” active subscriptions based on their interval
    - Logs a new transaction for each recurring payment
    - Output can be a console log or stored in memory

3. LLM Integration

Use a LLM to generate:

- Tags from `campaignDescription`
- Short summary of the campaign

Example LLM Output:

```
{
  "tags": ["disaster relief", "clean water", "Nepal"],
  "summary": "This campaign provides emergency aid to earthquake-affected communities in Nepal."
}
```

Stretch Goals

- Add a `GET /subscriptions` endpoint to list all active subscriptions
- Add a `GET /transactions` endpoint to view donation history
- Add support for multiple currencies and normalize amounts (e.g., in USD)
- Containerize using Docker