Dear Candidate,

Thank you for your interest in joining Dainin.

Please find the interview task below. This exercise is designed to help us understand how you think about systems, trade-offs, and real-world engineering problems.

Key details:

- Deadline: Wednesday 14th January 2026
- First interview calls: Friday 16th January 2026
- Submit completed tasks to: hr@dainin.co.uk

To accommodate time zone differences and keep the process fair for everyone, we will accept responses until Thursday mid-day (UK time).
Please note that while all submissions within this window will be reviewed, strong responses will be prioritised on a first-come basis.

At present, there is one open vacancy. However, if we receive two exceptionally strong submissions, we are open to hiring both candidates. Additionally, there may be opportunities for up to two intern roles if the responses demonstrate strong thinking but limited professional experience.

Payscale: As mentioned in the LinkedIn advert / communicated by the recruiter

We appreciate the time and thought this task requires and look forward to reviewing your response.

Best regards,
The Dainin Team

Full-Stack Systems & Judgment Challenge

(Yes, this is meant to be interesting)

We build real-world SaaS systems that deal with **payments, automations, AI, scale, and things breaking at 2am**.
This exercise helps us understand how you think when systems get messy.

Time: ~45–60 minutes
Format: Written answers (plain text only)
Next step: A conversation where we dig into *your* reasoning

A few friendly ground rules

- Plain text only — no code, diagrams, or screenshots
- There are no trick questions
- Trade-offs > perfection
- Clarity > length
- If something "depends", say what it depends on

You're welcome to use references or AI tools — just make sure the final answer reflects **your own thinking**. We care far more about judgment than polish.

Part 1: Understanding the Real Problem

Imagine a service-based SaaS business doing ~£30k/month.

Their setup includes:

- A CRM
- A calendar / scheduling tool
- Email and LinkedIn for lead generation
- Several automations built over time by different people

What they say:

- "Things break randomly"
- "No one fully understands the system"
- "Small changes cause unexpected issues"

Questions

1. What do you believe the *real underlying problem* is here?
   (Not the symptoms.)
2. Before writing a single line of code, what steps would you take to stabilise this system?
3. What's one solution that *sounds sensible* but would likely make things worse long-term? Why?

Part 2: System Placement & Architectural Judgment

Below is a description of a framework used in some SaaS products.

Framework description

- Maintains long-lived state
- Coordinates asynchronous workflows across multiple external services
- Handles retries, idempotency, and failure recovery
- Can resume execution after crashes without losing progress
- Enforces deterministic execution
- Not designed for low-latency request/response flows
- Assumes business processes may run for minutes, hours, or days

Questions

4. Where would this framework logically sit within a SaaS product? What role does it play in the system?
5. What would go wrong if someone tried to use this framework as:
    - a) a synchronous API request handler
    - b) a frontend-facing service
6. If this framework were *integral* to the product, what other system capabilities would need to exist around it for reliability?
7. What production signals or symptoms would suggest this framework is being misused?

Part 3: Pick Any Two Scenarios Below

(Choice is intentional — we want to see what you gravitate towards and how deeply you reason.)

A) Webhooks in the Real World

You're integrating with a payment provider. They send webhooks for successful payments, failures, refunds, and subscription changes.

Answer **any two**:

- A webhook arrives before your database has saved the order. What do you do?
- The provider goes down for 2 hours, then sends 10,000 webhooks in 5 minutes. What breaks in a naive system?
- You receive the same webhook 3 times and accidentally charge the user 3x. How do you prevent this?
- Six months later, users report "payment succeeded but account not upgraded". How do you diagnose this?

B) LLMs in Production (Where Things Get Weird)

You're building a document Q&A feature using an LLM API.

Answer **any two**:

- Your LLM costs jump from $50/day to $2,000/day. What options do you have to reduce costs without killing the feature?
- Users complain responses feel slow (8–12 seconds). Do you implement streaming? What complexities come with it?
- One prompt update made answers worse for 3 days before anyone noticed. How do you manage prompt changes safely?
- Your LLM provider goes down during peak hours. What's your resilience strategy?

C) Scaling & Data Judgment

Multi-tenant SaaS: agencies create AI chatbots for clients.

Answer **any two**:

- How would you handle multi-tenancy (separate DBs, schema-per-tenant, shared tables)? Why?
- You expect 100M+ messages. How do you store, partition, or archive them?
- An agency requests full GDPR deletion, but their data is deeply referenced. What makes this hard?

- Token-based billing must be accurate. How do you avoid overcharging or undercharging?

Part 4: Judgment & Trade-Offs

8. If the framework described earlier had to be removed entirely:

- What would break immediately?
- What would degrade slowly over time?

Explain your reasoning.

Final Notes

There's no "perfect" answer here.
We're looking for:

- how you reason
- how you prioritise
- how you think about failure
- and whether you've actually lived with systems like this

Strong answers usually mention trade-offs, uncertainty, and things that went wrong in the past.

Deadline: Wednesday 14th
First interview calls: Friday 16th January 2026

Thanks for taking the time — we know this isn't a small ask, and that's intentional. If you enjoyed thinking through this, you'll probably enjoy working with us too.

Best Wishes

Priya S

People and Process Manager

Dainin Group of Companies