Urumi AI SDE internship - Round 1

Stipend: 1,00,000 INR/pm + PPO possibility
Mode: Hybrid (1 month in Goa (paid) + Remote)
Duration: 6 months

Hiring Process

- **Round 1:** System Design
- **Round 2:** Gen AI Orchestration
- **Round 3:** Interview with Founders

Role Context

This internship role is **AI-focused**, but we also care deeply about **DevOps fundamentals** and **how fast you can learn and ship**.

Round 1 evaluates system design + Kubernetes orchestration. Round 2 builds on the same infrastructure using Gen AI.

This task is difficult, but you are going to learn a lot.

Don't submit AI-generated work you don't understand. During the demo video and interview, we will ask you to walk through your solution end-to-end and defend your choices.

Disclaimer

- The work you'll do as part of this test task will be completely yours and you will own the copyright to that. Urumi will not use the test task code in production.

System Design Assessment — Round 1: Kubernetes Store Orchestration (Local-to-Prod)

AI usage (allowed, expected, but you must understand internals)

You are encouraged to use AI tools to speed up implementation. However:

- You must be able to **explain what you built** and **why it works**.
- You should understand the internals of what's happening (Kubernetes resources, orchestration flow, persistence, ingress, isolation).
- Assume we may ask follow-ups on architecture, reliability, and scaling.

Problem statement

Build a small "store provisioning platform" that runs on local Kubernetes but is designed so the **same Helm-based deployment** can run in production (for example, on a VPS box running k3s) with **configuration changes only**.

User story

As a user, I should be able to:

1. Open a **Node Dashboard** (React web app).
2. View existing stores and their status.
3. Click **Create New Store**.
4. The system provides a **functioning ecommerce store** automatically.
5. I can provision **multiple stores** concurrently.
6. Each store can be either:
    - **WooCommerce (WordPress + WooCommerce)**, or
    - **MedusaJS**
7. The dashboard shows for each store:
    - status (Provisioning / Ready / Failed)
    - store URL(s)
    - created timestamp
8. I can **delete** a store and all resources are cleaned up.

Definition of Done (must be testable)

A provisioned store must support placing an order end-to-end. Any default storefront/theme is fine.

WooCommerce

- Open storefront
- Add a product to cart
- Checkout using a test-friendly method (COD/dummy gateway is fine)
- Confirm order is created (visible in WooCommerce admin)

Medusa

- Open storefront (default starter is fine)
- Add a product to cart
- Complete checkout supported by the starter
- Confirm the order exists (via admin UI or API)

Scope note: You may fully implement **either WooCommerce or Medusa** in Round 1. The other can be stubbed, but your architecture should make adding the second engine straightforward.

Kubernetes + Helm requirements (mandatory)

- Must run on local Kubernetes (Kind / k3d / Minikube).
- Must be deployable to a production-like VPS setup (k3s is fine) using **the same Helm charts**.
- **Helm is mandatory** (no Kustomize). Local vs production differences must be handled via **Helm values** (e.g., `values-local.yaml`, `values-prod.yaml`).
- Provisioning must be Kubernetes-native (Deployments/StatefulSets, Services, Ingress, PVCs, Secrets, Jobs/Controllers).

- Multi-store capability with isolation (namespace-per-store preferred).
- Persistent storage for the database at minimum.
- Each store exposed via HTTP using Ingress with stable URLs (document your local domain approach).
- Basic readiness/liveness checks.
- Clean teardown: deleting a store removes its resources safely.
- No hardcoded secrets in source code.

Demo Video + Submission (mandatory)

Submission (via form)

Please submit:

- **Demo Video**
- **GitHub Repo**
- **Form URL: https://dashboard.urumi.ai/s/roundoneform2026sde**
- **Form will strictly close by 13 February 2026, 11:59 PM IST and no further submissions will be accepted**

Demo Video requirements

In your demo video, please cover the demo of the actual thing you've built, covering:

1. **System design & implementation**
    - components and responsibilities
    - end-to-end flow: create store → resources created → store ready → place order → delete store
2. **Isolation, resources, reliability**
    - how stores are isolated (namespaces, secrets, PVCs)
    - requests/limits and any quotas/guardrails

    - idempotency and failure handling
    - cleanup guarantees
3. **Security posture**
    - secret handling
    - RBAC / least privilege approach
    - what's exposed publicly vs internal-only
    - basic container hardening (where applicable)
4. **Horizontal scaling plan**
    - what scales horizontally (API, orchestrator, dashboard)
    - how you'd scale provisioning throughput
    - stateful constraints and how you'd handle them
5. **Abuse prevention**
    - rate limiting / quotas (per user or per IP)
    - blast-radius controls (max stores, max resources, timeouts)
    - audit trail/logging for actions
6. **Local-to-VPS production story**
    - what changes between local and VPS via Helm values
    - ingress, domains, storage, secrets strategy
    - upgrade/rollback approach with Helm

Deliverables (in the repo)

- `README.md` including:
    - local setup instructions

    - VPS/production-like setup instructions (k3s is fine)
    - how to create a store and place an order
- Source code for dashboard + backend + provisioning/orchestration
- Helm chart(s) + values files (local vs prod)
- Short “System design & tradeoffs” note covering:
    - architecture choice
    - idempotency/failure handling/cleanup approach
    - what changes for production (DNS, ingress, storage class, secrets, etc.)

Ways to Stand Out

These will be strong differentiators if implemented well.

1) Production-like VPS deployment

- Deploy the same Helm charts on a **VPS (k3s)** and show it working.
- Document what changed via Helm values (domain/ingress/storage/secrets).
- Optional: TLS (cert-manager) setup notes.
- Optional: Add in dashboard for people to link their domain to the store.
- Deploy live on AWS or google free tier.

2) Stronger multi-tenant isolation and guardrails

- Namespace-level **ResourceQuota** + **LimitRange** per store.
- Optional: max PVC size per store; default requests/limits for pods.

3) Idempotency and recovery

- Store creation is safe to retry (no duplicate resources, clean reconcile).
- If the provisioning component restarts mid-provisioning, the system recovers or fails cleanly with clear status.

4) Abuse prevention beyond rate limiting

- Per-user/store quotas (e.g., max stores per user).
- Timeouts for provisioning.
- Audit log: who created/deleted what and when.

5) Observability

- Store-level events or a small activity log surfaced in the dashboard.
- Basic metrics (e.g., stores created, provisioning failures, provisioning duration).
- Clear “why it failed” reporting when something breaks.

6) Network and security hardening

- Basic RBAC with least privilege for provisioning component.
- NetworkPolicies per store namespace (deny-by-default with required allows).
- Run containers as non-root where possible.

7) Scaling plan (implemented, not just described)

- Horizontal scaling for platform components (API/dashboard/orchestrator).
- Concurrency controls for provisioning multiple stores safely.

8) Upgrades and rollback story

- Demonstrate or document how you’d upgrade store versions (images/chart values) and roll back safely with Helm.

Round 2: Gen AI Orchestration (details shared later)

In Round 2, you will use the infrastructure you built in Round 1 to **enable store creation via Gen AI orchestration**. Details and constraints will be shared at the start of Round 2 for shortlisted candidates.