



DevSprint 2026

IUT Computer Society

1. The Scenario: The IUT Cafeteria Crisis

At the peak of Ramadan, the IUT Cafeteria faces a legendary digital rush. As 5:30 PM approaches, hundreds of fasting students stare at their phones, thumbs ready over the “Order Now” button. When ordering opens, the aging “Spaghetti Monolith” begins to choke. Database locks create bottlenecks, requests start timing out, and while the frontend shows a calm loading spinner, the backend struggles with deadlocked threads and failing services.

Last week, things escalated: the Ticketing Service crashed completely. The kitchen received no orders, and students were left confused as their orders seemed to disappear into the system. Long physical queues formed as the server froze under the heavy load.

It’s now clear that a single-server monolith cannot handle the surge of hundreds of hungry engineers. The university administration has called for a complete architectural overhaul. Your mission is to break this fragile monolith into a distributed, fault-tolerant microservice system; one where a failure in the Notification Service won’t leave a student’s Iftar stuck in digital limbo, and the system can survive the Ramadan rush reliably.

2. System Architecture

You are required to build and containerize the following services. Each must be isolated in its own environment and communicate over the network.

Service	Core Responsibility	Key Functionality
Identity Provider	Authentication & Authorization	Single source of truth for student verification; issues secure JWT tokens.
Order Gateway	Primary Entry Point	API Gateway that performs mandatory Token Validation and a High-Speed Cache Stock Check before hitting the database, reducing load.
Stock Service	Inventory Management	The "Source of Truth" for inventory. Must use robust Concurrency Control (e.g., Optimistic Locking) to ensure transactional stock decrement and prevent over-selling.
Kitchen Queue	Asynchronous Order Processing	Immediately acknowledges successful orders (<2s). Decouples user feedback from the time-intensive cooking/preparation process (3-7s).
Notification Hub	Real-Time Communication	Push instantaneous order status updates (e.g., <i>Confirmed</i> , <i>Ready</i>) directly to the student UI, eliminating client polling.

N.B: Judges should be able to run the whole system using a single `docker compose up` command

3. Core Engineering Requirements

A. Security & Authentication

- **Token Handshake:** The client must authenticate with the Identity Provider to receive a secure token.
- **Protected Routes:** The Order Gateway must reject any request missing a valid bearer token with a 401 Unauthorized status.

B. Resilience & Fault Tolerance

- **Idempotency Check:** You must design for partial failures where a service performs an action (like deducting stock) but crashes before sending a success response.
- **Asynchronous Processing:** To maintain responsiveness, the Kitchen Service must decouple the "acknowledgment" from the "execution".

C. Performance & Caching

- **Efficient Caching:** Implement a caching layer in front of the Stock Service. During peak rush, the Gateway should check the cache first; if the cache reports zero stock, the request must be rejected instantly to protect the database from unnecessary load.

D. Automated Validation (CI/CD)

- **Integrity Testing:** You must write unit tests for Order Validation and Stock Deduction logic.
- **Automated Pipeline:** Every push to the main branch must trigger a pipeline that runs these tests. The build must fail if any test fails, ensuring no broken code reaches the production environment.

4. Observability & Monitoring

Every service must expose the following:

- **Health Endpoints:** A route returning 200 OK if the service and its dependencies are reachable, or 503 Service Unavailable if a dependency is down.
- **Metrics Endpoints:** Machine-readable data tracking total orders processed, failure counts (500-errors/timeouts), and average response latency.

5. Interface Requirements

Student Journey UI

A single-page application demonstrating the full user flow:

1. **Authentication:** Secure login to obtain a token.
2. **Order Placement:** Authenticated trigger for the Iftar flow.
3. **Live Status:** A real-time tracker showing the order transition: Pending → Stock Verified → In Kitchen → Ready.

Admin Monitoring Dashboard

A separate view to visualize the system's operational health:

- **Health Grid:** Visual indicators (Green/Red) for each microservice.
- **Live Metrics:** Real-time display of latency and order throughput.
- **Chaos Toggle:** A manual trigger to "kill" a service, allowing judges to observe how the UI and remaining services handle partial failure.

6. Bonus Challenges

- **Cloud Frontier:** Deploy the entire containerized ecosystem to a cloud provider of your choice.
- **Visual Alerts:** Implement a monitoring system that triggers visual alerts if the Gateway's average response time exceeds 1 second over a 30-second window.
- **Rate Limiting:** Protect the Identity Provider. Implement a rate-limiter that restricts a single Student ID to 3 login attempts per minute to prevent brute-force attacks or script-based "botting" of orders.