ER Triage System – End-to-End Logic and Flow

This document describes the complete logical flow of the Emergency Room (ER) Triage System, showing how all components—UI, controllers, services, DAOs, triage logic, and database—interact together during real-world operations.

1. Overview

This example demonstrates how data flows through the system from the moment a nurse enters patient information to when it is saved, evaluated, displayed, updated, and finalized in the database.

2. Scenario Steps

We will walk through an example involving two patients (Noura and Ali):

1. Nurse intakes Noura (Level 2 → WAITING)
2. Queue listing shows Noura
3. Noura's vitals worsen → automatic escalation to Level 1 (IN_TREATMENT)
4. Nurse intakes Ali (Level 3 → WAITING)
5. Queue listing now shows only Ali
6. Noura is marked as TREATED

3. Detailed Flow

Step 1: Intake – Create Patient (Noura)

The nurse enters Noura's details into the intake form. The frontend sends this data as a CreatePatientRequest to POST /api/patients. The PatientController passes this request to PatientService.createPatient(req). PatientService calls ValidationService to ensure data validity, then KeywordService to extract relevant terms from symptom text. It constructs a Patient object with Vitals and calls TriageService.evaluate(patient) to calculate triage level and score. Since no red flag is detected, Noura is marked as WAITING (Level 2).

PatientDAO.save(patient) stores the record in the database. AuditService records this creation event. A PatientView is returned to the UI for confirmation.

Database after save:
• ID: 1 | Name: Noura | Age: 70 | Level: 2 | Score: 8 | Status: WAITING | Reason: SBP 102; SpO_2 93%; chest pain

Step 2: Queue Listing

The nurse requests the waiting queue using GET /api/patients?status=WAITING. PatientController calls QueueService.listWaiting(), which queries the database for all patients with status='WAITING' and orders them by triage level, score, time, and age. The result is converted into PatientView objects and displayed on the queue page. At this point, Noura is the only waiting patient.

Step 3: Update Vitals – Escalation to Level 1

Later, Noura's condition worsens (SpO_2 84, RR 36). The nurse sends PATCH /api/patients/1/vitals with the new vitals. PatientService.updateVitals() fetches the existing record, validates the update, applies the new readings, and re-evaluates her using TriageService. The system detects a red flag and automatically changes her status to IN_TREATMENT. PatientDAO.update(patient) saves this change, and AuditService logs the escalation event.

Database after save:
• ID: 1 | Name: Noura | SpO_2: 84 | RR: 36 | Level: 1 | Status: IN_TREATMENT | RedFlag: true

Step 4: Intake – Create Patient (Ali)

Another patient, Ali, arrives. His vitals are normal (SpO_2 97, BP 120/80, HR 88, RR 18). The system runs through the same validation and triage process, resulting in Level 3 (low urgency). He is saved as WAITING and logged in the audit table.

Database now contains:
• Noura – Level 1, IN_TREATMENT
• Ali – Level 3, WAITING

Step 5: Queue Listing (Ali only)

The nurse reloads the queue. QueueService.listWaiting() again fetches only patients with status='WAITING'. This time, only Ali is displayed.

Step 6: Change Status – Noura Treated

After treatment, the doctor updates Noura's status to TREATED using PATCH /api/patients/1/status. PatientService.changeStatus() validates the transition (IN_TREATMENT → TREATED), updates the database, and records an audit entry. Noura is no longer part of active lists but remains in system history for reporting.

4. Function Relationships

Each backend function participates in these flows as follows:
• PatientController.createPatient → PatientService.createPatient → ValidationService → KeywordService → TriageService → PatientDAO.save → AuditService.record
• PatientController.patchVitals → PatientService.updateVitals → TriageService.evaluate → PatientDAO.update → AuditService.record

• PatientController.listWaiting → QueueService.listWaiting → PatientDAO.listWaiting
• PatientController.patchStatus → PatientService.changeStatus → ValidationService.validateStatusChange → PatientDAO.update → AuditService.record

5. Overall Logic Summary

1. Every patient creation or update flows through the PatientService.
2. The TriageService determines level, score, and red flag conditions.
3. Red flags automatically change patient status to IN_TREATMENT (bypass queue).
4. QueueService only lists WAITING patients, ordered by urgency.
5. All changes are recorded through AuditService for traceability.
6. The database reflects real-time triage state for both treatment and reporting.

6. Key Takeaway

The system's architecture ensures medical realism and logical integrity. Each update triggers re-evaluation and automatic priority adjustment, while all changes are safely logged and version-controlled. This mirrors real ER triage workflow where data entry, decision-making, and queue management are seamlessly connected.