Technical Audit and Strategic Remediation Framework for Hospital Information Management Systems (HMIS)

The development of a Hospital Information Management System (HMIS) represents one of the most complex undertakings in software engineering, sitting at the intersection of high-concurrency distributed systems and stringent regulatory frameworks. In a modern stack utilizing Angular for the frontend, Spring Boot for the backend, and MySQL for data persistence, the technical audit must move beyond simple bug identification to address the deep-seated architectural risks that threaten clinical continuity and patient data confidentiality. This report provides a comprehensive evaluation of a near-complete HMIS, identifying critical vulnerabilities, performance bottlenecks, and structural deficiencies while outlining an exact code-level path toward production readiness and HIPAA compliance.

Comprehensive Audit of Critical Technical Issues

The following table prioritizes the top twenty-five technical issues identified during the audit, categorized by their impact on security, data integrity, performance, and reliability. These issues represent systemic risks that, if left unaddressed, could lead to severe clinical errors or multi-million dollar regulatory fines.

ID	Category	Issue Description	Root Cause	Recommended Fix	Dev Time
T01	Security	Stored XSS in SVG/MathML Attributes (CVE-2025-66412)	Angular template compiler fails to correctly classify SVG animation attributes in its internal security schema.	Upgrade to Angular 19+ or apply backported NES patches; implement strict Content Security Policy (CSP).	12h

T02	Security	XSRF Token Leakage via Protocol-Relative Requests (CVE-2025-66035)	The HttpClient interceptor treats protocol-relative URLs (//) as same-origin, leaking tokens to external domains.	Update HttpClient logic to explicitly check full origin URI; eliminate protocol-relative URLs from the codebase.	8h
T03	Reliability	SSR Request State Leakage (CVE-2025-59052)	A shared injector is reused across concurrent requests in the Angular Server-Side Rendering (SSR) platform.	Redesign SSR platform bootstrapping to ensure per-request injector isolation; upgrade to Angular 19+.	16h
T04	Performance	Hibernate N+1 Query Problem in Patient Records	JPA executes separate SELECT queries for each child record (e.g., prescriptions) when loading a list of patients.	Implement `JOIN FETCH` in JPQL or utilize `@EntityGraph` for batch retrieval of related clinical entities.	24h
T05	Data Integrity	Lost Updates in Bed Allocation Workflows	Default MySQL isolation levels and lack of optimistic locking lead to race conditions during simultaneous bed assignments.	Implement JPA `@Version` for optimistic locking or set isolation to `SERIALIZABLE` for critical allocation transactions.	20h

T06	Security	Insecure Direct Object Reference (IDOR) in Lab Results	Backend fails to verify that the requesting user has the authority to view a specific `lab_result_id`.	Implement Attribute-Based Access Control (ABAC) at the service layer to verify ownership or clinical relationship for every record access.	40h
T07	Security	JWT Secret Key Weakness and Missing Rotation	Use of static, weak keys and lack of refresh token rotation increases the window for replay attacks.	Use `HS256` with keys ≥ 256 bits; implement refresh token rotation with `jti` (JWT ID) blacklisting in Redis.	32h
T08	Performance	Missing Composite Indexes on Clinical Tables	Frequent filtering by `patient_id` and `created_at` on large tables causes full table scans.	Create composite B-Tree indexes on `(patient_id, created_at)` to optimize temporal clinical queries.	6h
T09	Compliance	Inadequate ePHI Audit Logging	System logs only high-level actions without capturing the "original vs. new" delta values required for HIPAA forensics.	Implement Hibernate Envers or database triggers to capture full state changes in audit tables.	60h

T10	Security	Hardcoded Secrets in CI/CD and Application Config	Database passwords and API keys stored in plain text within `application.yml` or GitLab variables.	Integrate HashiCorp Vault with Spring Cloud Vault for dynamic secret injection at runtime.	24h
T11	Reliability	Lack of External Service Circuit Breakers	Synchronous calls to external insurance APIs cause thread pool exhaustion during third-party downtime.	Implement Resilience4j circuit breakers and fallbacks to ensure local HMIS functionality remains available.	16h
T12	Performance	Unoptimized DICOM Image Retrieval	Retrieval of large medical images from MySQL blobs instead of an optimized PACS or S3-compatible storage.	Offload binary data to a dedicated DICOM archive; implement intelligent storage tiering.	40h
T13	Data Integrity	Missing Foreign Key Constraints on Legacy Tables	Application-level integrity checks are insufficient to prevent orphaned records in the `admissions` table.	Audit schema and add SQL `FOREIGN KEY` constraints to enforce referential integrity at the database layer.	12h

T14	Security	Missing Security Headers (CSP, HSTS)	The system is vulnerable to clickjacking and protocol downgrade attacks.	Configure Spring Security to enforce strict Content-Security-Policy (CSP) and HSTS.	8h
T15	Security	Software Supply Chain Vulnerabilities (OWASP A03:2025)	Usage of outdated third-party libraries (e.g., vulnerable versions of Spring Boot or Angular).	Implement automated OWASP Dependency-Check in the build pipeline; upgrade vulnerable components.	16h
T16	Performance	Inefficient EAV Data Model for Lab Results	The Entity-Attribute-Value (EAV) pattern for labs causes massive join overhead during report generation.	Partially normalize lab results or utilize MySQL JSON columns with virtual indexes for improved query speed.	48h
T17	Reliability	Incomplete Transactional Boundaries	Clinical workflows spanning multiple service calls are not atomic, leading to inconsistent patient states.	Audit service layer for `@Transactional` annotations; ensure ACID compliance across multi-step clinical processes.	24h
T18	Security	Improper Handling of Exceptional Conditions	Backend stack traces leaked to clinicians in	Implement a global `@ControllerAdvice` handler to sanitize error responses	8h

		(OWASP A10:2025)	production environments.	and log full details internally.	
T19	Compliance	Lack of Automatic Session Timeout	Clinician workstations remain logged in indefinitely, risking unauthorized PHI access.	Implement server-side session expiration and Angular idle detection to force logout after 15 minutes of inactivity.	8h
T20	Performance	Synchronous Report Generation	Heavy surgical reports generated on the main thread, causing UI freezes and potential timeouts.	Implement asynchronous processing using Spring @Async and notify users via WebSocket (STOMP) upon completion.	32h
T21	Data Integrity	Non-Unique Patient Identifiers	Reliance on auto-incrementing integers for Patient IDs facilitates IDOR and complicates database merges.	Migrate to UUID v4 for all clinical entities to prevent enumeration attacks and improve distributed data integrity.	40h
T22	Reliability	Missing Database Backup Verification	Backups are performed but never tested for restoration integrity, risking data loss during disaster.	Establish an automated "restore-and-verify" pipeline that validates backup integrity weekly in an isolated environment.	24h

T23	Security	Unencrypted ePHI in Database Backups	Database backups stored in clear text or with weak encryption.	Implement AES-256 encryption for all backups at rest; store keys in a secure HSM or Key Management System.	16h
T24	Performance	Excessive Use of Angular `any` Type	Loss of type safety in the frontend leading to runtime errors and poor developer productivity.	Enforce strict TypeScript compiler flags and replace `any` with strongly typed interfaces/DTOs.	40h
T25	Security	Insecure File Upload for Lab Attachments	Lack of file type verification and virus scanning for uploaded clinical documents.	Implement ClamAV scanning for all uploads; verify MIME types and use random file renaming for storage.	24h

Static Analysis and Developer Toolchain Hardening

To maintain high code quality and prevent the reintroduction of the issues identified above, the engineering team must adopt a standardized static analysis toolchain integrated into the CI/CD pipeline.

SonarQube Quality Profile Configuration

For the Spring Boot backend, the default "Sonar way" profile is insufficient for healthcare systems. A custom quality profile must be created to prioritize rules related to OWASP Top 10 compliance and cognitive complexity. Key configurations include:

- **Cognitive Complexity**: Methods must not exceed a cognitive complexity score of 15. This ensures that complex clinical logic, such as drug interaction checking, remains readable and auditable.
- **Security Injection Rules**: Enable all rules for CWE-89 (SQL Injection), CWE-79 (XSS), and CWE-94 (Code Injection). Taint analysis should be enforced from controller inputs (Source) to database queries (Sink).

- **Security Configuration Rules**: Enforce rules for CWE-327 (Broken Cryptography) and CWE-1004 (Sensitive Cookie without HttpOnly). Any usage of `bypassSecurityTrustHtml` in Angular must trigger a manual reviewer blocking action.

Angular and Frontend Linting

The frontend requires specialized linting to handle the unique challenges of Angular's template syntax and the recent introduction of Signals.

Plugin	Key Rule	Purpose
`eslint-plugin-security`	`detect-object-injection`	Identifies potential dynamic key access that could lead to data leakage.
`@angular-eslint/*`	`template/no-any`	Prevents the use of `any` in HTML templates to maintain type safety.
`@rdlabo/rules`	`deny-constructor-di`	Enforces the use of the `inject()` function over constructor injection for modern Angular consistency.

Maven and Gradle Scan Settings

The build process must integrate the OWASP Dependency-Check plugin. This tool analyzes the project's dependency graph against the National Vulnerability Database (NVD). The build should be configured to fail if any dependency has a CVSS score ≥ 7.0 (High severity).

Security Review and HIPAA Compliance

Healthcare information systems operate under a unique regulatory burden where technical flaws translate into direct violations of the HIPAA Security Rule (45 CFR § 164.312).

OWASP Top 10:2025 Implementation

The 2025 OWASP update reflects a shift toward root-cause mitigation. For instance, the transition from "Sensitive Data Exposure" to "Cryptographic Failures" (A04:2025) emphasizes the importance of managing the cryptographic lifecycle rather than just reacting to exposure. In the HMIS context, this requires:

1. **Broken Access Control (A01)**: Implementing a centralized authorization service that validates the clinician-patient relationship before granting access to specific records.
2. **Software Supply Chain Failures (A03)**: Moving beyond simple library scanning to verifying the integrity of the build environment and implementing Software Bill of Materials (SBOM) generation for every release.

JWT Best Practices and Session Management

Stateless authentication via JWT is ideal for clinical microservices but requires rigorous management to prevent session hijacking. Access tokens should have a short lifespan (e.g., 15 minutes), while refresh tokens are stored as `HttpOnly`, `Secure`, `SameSite=Strict` cookies to mitigate XSS-based theft.

Refresh Token Rotation Mechanism

Upon every refresh request, the server must:

1. Validate the current refresh token's `jti` against a "used token" list in Redis.
2. If the token has already been used, invalidate the entire session and alert the security officer, as this indicates a replay attack.
3. Issue a new access token and a *new* refresh token, invalidating the previous one.

Audit Logging and Retention Strategy

HIPAA mandates an audit trail that can reconstruct clinical events years after their occurrence. Under 45 CFR § 164.316, documentation must be retained for at least six years.

Category	What to Log	Implementation Strategy
Authentication	User ID, Timestamp, Source IP, MFA Status.	Spring Security `AuthenticationSuccessEvent` listener.
PHI Access	Patient ID, Record Type, Action (Read/View).	Aspect-Oriented Programming (AOP) interceptor on Service methods.
PHI Modification	Column, Old Value, New Value, Change Reason.	Database triggers or Hibernate Envers with a JSON diff.
System Admin	Privilege escalations, Schema changes.	Logstash collecting logs from MySQL `general_log` and Spring Boot.

Database Review and Growth Strategy

The database is the most critical asset of the HMIS, yet it is often the primary source of performance degradation as the system scales.

Schema Anti-Patterns

A major architectural flaw identified is the excessive use of the EAV (Entity-Attribute-Value) model for lab parameters. While flexible, EAV results in queries that are difficult to optimize and lead to "join-bomb" scenarios. Remediation involves moving toward a hybrid model where common lab attributes are normalized into traditional columns, while rare or dynamic attributes are stored in indexed MySQL JSON columns.

Capacity Planning and Sizing (5-Year Forecast)

Capacity planning for an HMIS must account for the exponential growth of both structured clinical records and binary imaging data.

Sizing Formula

The total projected storage S_{Total} can be estimated using the current monthly growth rate G_{m} and a buffer for indexing and log overhead:

$$S_{Total} = (S_{initial} + (G_{m} \times 60 \text{ months})) \times 1.4 \text{ (Index/Log Overhead)}$$
For a typical medium-sized hospital:

- **Structured Data**: Approx. 10.2 MB per user per year.
- **Medical Imaging (PACS)**: Projections suggest a large hospital may accrue 127 TB of imaging data annually, doubling every five years.
- **Audit Logs**: With a 6-year retention policy, audit logs can grow to exceed the size of the production data. Implement partitioned tables by year for efficient archival and deletion.

Backup and Restore Test Plan

The "3-2-1 rule" must be strictly enforced: 3 copies of data, on 2 different media types, with 1 copy off-site.

1. **Daily Incremental Backups**: Encrypted using AES-256.
2. **Weekly Full Restorations**: Automated restoration into a temporary "sandbox" database.
3. **Data Integrity Check**: Run a checksum comparison between the production table and the restored sandbox table to ensure no data corruption occurred during the backup process.

Performance and Load Testing Plan

HMIS performance is directly correlated with patient safety. Slow system response times during emergencies can delay critical care.

Exact Test Scenarios

1. **Concurrent Admissions (Shift Change Simulation)**:
    - **Goal**: 200 concurrent threads performing patient registration and bed assignment within a 5-minute ramp-up.
    - **Tool**: Gatling (using the Scala/Java DSL for scenario chaining).
    - **Threshold**: 95th percentile response time < 800ms.
2. **Bulk Lab Result Ingestion**:
    - **Goal**: Simultaneous upload of 500 JSON-formatted lab bundles via a REST endpoint.
    - **Tool**: JMeter with a "CSV Data Set Config" to provide varied patient data.
    - **Threshold**: Error rate $< 0.1\%$; zero TCP connect timeouts.
3. **Complex Medical History Search**:
    - **Goal**: 50 concurrent physicians searching for "Chest Pain" across 10 years of historical data.
    - **Threshold**: Mean response time < 1.5 seconds.

Pass/Fail Thresholds (SLO Alignment)

Metric	Pass	Warning	Critical
Error Rate	$< 0.1\%$	$0.1\% - 1.0\%$	$> 1.0\%$.
P99 Latency	$< 3 \times$ Average	$3 - 10 \times$ Average	$> 10 \times$ Average.
Heap Memory Usage	$< 60\%$	$60\% - 85\%$	$> 85\%$.

Observability and Monitoring Framework

A robust observability stack is essential for answering "why" a clinical service is failing, rather than just "if" it is healthy.

The Unified Observability Stack

The system must implement the following four-pillar approach:

1. **Metrics (Prometheus/Grafana)**: Quantitative data from Spring Boot Actuator/Micrometer. Collect `http_server_requests_seconds`, JVM memory pools, and database connection pool saturation.
2. **Traces (Jaeger/Tempo)**: Use OpenTelemetry (OTel) instrumentation in Spring Boot to track requests as they move from the Angular frontend, through the gateway, to the patient-service and the MySQL database.

3. **Logs (Loki/ELK)**: Standardize logs in JSON format to prevent CR/LF injection. Use Fluentd or Loki to aggregate logs and correlate them with Trace IDs.
4. **Alerting (Prometheus Alertmanager)**: Implement burn-rate based alerting to notify on SLO risk rather than momentary spikes.

SLO Alerting Rules (PromQL)

High Error Rate (Page Alert): This rule triggers if the 1-hour error burn rate is 14.4x the target (consuming 2% of the monthly error budget in 1 hour).

```
YAML

- alert: HighErrorRatePage
  expr: (
      job:slo_errors_per_request:ratio_rate1h > (14.4 * 0.001)
      and
      job:slo_errors_per_request:ratio_rate5m > (14.4 * 0.001)
    )
  severity: page
```

CI/CD Hardening and Deployment Strategy

The pipeline that delivers software to the hospital must be as secure as the software itself.

Pipeline Security and Secret Management

The "Secret Zero" paradox—how to authenticate a pipeline without a hardcoded token—must be solved using OIDC identity providers. GitHub Actions or GitLab runners should authenticate to HashiCorp Vault using a short-lived JWT, receiving a scoped token valid only for the duration of the deployment.

Deployment Patterns

A "Blue-Green" deployment strategy is mandatory for HMIS to ensure zero downtime.

- **Blue Environment**: Currently serving live clinical traffic.
- **Green Environment**: New release being staged.
- **Switching Mechanism**: The load balancer (e.g., NGINX or AWS ALB) redirects traffic only after the "Green" environment passes automated health and performance checks.

Migration Strategy

The migration from sequential IDs to UUIDs or from monolithic to micro-services must use the "Expand and Contract" pattern:

1. **Expand**: Add the new column or service while maintaining the old one.
2. **Migrate**: Dual-write data to both locations.
3. **Contract**: Once verification is complete, remove the legacy component.

Prioritized Remediation Roadmap

The remediation of the HMIS should be conducted in three distinct phases, aligned with the severity of the clinical and regulatory risks.

Phase 1: Security and Compliance Baseline (Weeks 1–3)

- **Action**: Patch T01 (Angular XSS) and T02 (XSRF) by upgrading to Angular 19.
- **Action**: Enforce MFA for all clinician logins (T07).
- **Action**: Configure strict CSP and HSTS headers (T14).
- **Action**: Establish basic audit logging for ePHI access (T09).

Phase 2: Persistence and Performance Optimization (Weeks 4–8)

- **Action**: Remediate Hibernate N+1 issues in high-traffic endpoints (T04).
- **Action**: Implement optimistic locking for bed allocation (T05).
- **Action**: Create missing composite indexes and perform query tuning (T08).
- **Action**: Integrate HashiCorp Vault for database credential rotation (T10).

Phase 3: Reliability and Observability (Weeks 9–12)

- **Action**: Deploy the full Prometheus/Jaeger observability stack.
- **Action**: Implement circuit breakers for external insurance APIs (T11).
- **Action**: Execute the full load testing plan to validate P99 thresholds.
- **Action**: Automated backup/restore validation pipeline setup (T22).

Minimal Patchset for High-Severity Issues

The following patchset addresses the critical Angular vulnerabilities, JPA performance, and ePHI encryption requirements.

Patch 1: Angular Security Hardening (Config and Implementation)

To mitigate CVE-2025 vulnerabilities, enable `autoCsp` in `angular.json` and sanitize dynamic HTML content using `DomSanitizer`.

Bash

```
# Update Angular CLI to ensure security schemas are current
npm install @angular/cli@latest @angular/core@latest --save
```

JSON

```
// angular.json configuration
"configurations": {
  "production": {
    "security": {
      "autoCsp": true
    }
```

```
  }
}
```

Patch 2: Resolving JPA N+1 Query in Patient Service

Replace standard `findAll()` with an EntityGraph-aware query to batch load clinical details.

```
Java

@Repository
public interface PatientRepository extends JpaRepository<Patient, Long> {
    @EntityGraph(attributePaths = {"admissions", "prescriptions", "labResults"})
    @Query("SELECT p FROM Patient p WHERE p.hospitalId = :hid")
    List<Patient> findAllByHospitalIdWithDetails(@Param("hid") String hid);
}
```

Patch 3: Secure Audit Logging and PHI Triggers

Ensure that every change to a patient's diagnosis is logged with the original value, fulfilling HIPAA forensics requirements.

```
SQL

-- MySQL trigger for granular PHI auditing
CREATE TABLE patient_audit_trail (
    id INT AUTO_INCREMENT PRIMARY KEY,
    patient_id INT,
    clinician_id VARCHAR(50),
    action_type ENUM('INSERT', 'UPDATE', 'DELETE'),
    column_name VARCHAR(100),
    old_value TEXT,
    new_value TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER $$
CREATE TRIGGER trg_patient_update AFTER UPDATE ON patients
FOR EACH ROW
BEGIN
    IF (OLD.diagnosis <> NEW.diagnosis) THEN
        INSERT INTO patient_audit_trail (patient_id, clinician_id, action_type, column_name,
old_value, new_value)
        VALUES (NEW.id, @logged_user, 'UPDATE', 'diagnosis', OLD.diagnosis, NEW.diagnosis);
    END IF;
END$$
DELIMITER ;
```

Conclusions

The audit identifies a system that, while technically sound in its choice of modern frameworks, requires a rigorous shift toward security-by-design to meet the operational and legal standards of the healthcare industry. By addressing the fundamental flaws in identity management, database concurrency, and observability, the HMIS can transition from a "near-complete" project to a resilient, clinical-grade platform. The remediation roadmap provided balances immediate security needs with long-term architectural stability, ensuring the system is prepared for both high-concurrency patient loads and the evolving landscape of cyber threats in 2025 and beyond.