LTU Health System - Technical Documentation

Module: COM7033 Secure Software Development
Version: 1.0.0
Date: 2025

---

1. Design Rationale & Architecture

The LTU Health System was architected to address the specific security and scalability challenges inherent in handling sensitive medical data (PII/PHI). The design follows the **Defense-in-Depth** principle.

1.1 Hybrid Polyglot Database Architecture

Instead of a monolithic database, the system utilizes a **Polyglot Persistence** pattern:

- **Authentication Layer (SQLite):**
    - **Why:** SQLite is transactional (ACID compliant) and highly efficient for structured, relational data like user credentials.
    - **Security Benefit:** By isolating credentials in a separate file (`auth.db`), we minimize the blast radius. A SQL Injection vulnerability in the auth layer does not inherently grant query access to the medical records stored in NoSQL.
- **Data Layer (MongoDB):**
    - **Why:** Medical records vary between patients (e.g., some have stroke history, others don't). MongoDB's schema-less nature allows for flexible data modeling without complex `JOIN` operations.
    - **Security Benefit:** MongoDB allows for document-level operations and pairs well with application-level encryption logic.

1.2 Security Controls Implementation

- **Confidentiality (Encryption at Rest):** All Personally Identifiable Information (PII)—specifically Names and NHS Numbers—are encrypted using **Fernet (AES-128)** before insertion into MongoDB. This ensures that a database dump is unreadable without the `secret.key`.
- **Integrity (Audit Logging):** Critical actions (Login, Update Record, Delete User) are written to an immutable `audit_logs` collection. This ensures non-repudiation and GDPR accountability.
- **Availability (Rate Limiting):** The `Flask-Limiter` extension enforces a limit of **10 requests per minute** on authentication endpoints to mitigate Brute Force and DoS attacks.
- **Input Validation:** The `clean_input()` wrapper utilizes the **Bleach** library to strip malicious tags, preventing Cross-Site Scripting (XSS).

---

2. Installation & Usage Instructions

2.1 Prerequisites

- **Python 3.8+** installed.
- **MongoDB Community Server** installed and running on `localhost:27017`.

2.2 Setup Steps

1. **Install Dependencies:**

```
pip install Flask pymongo fpdf Flask-WTF Flask-Limiter cryptography bleach
pandas openpyxl email_validator scikit-learn joblib
```

2. **Initialize System:** Run the setup script to generate encryption keys, create databases, and migrate the CSV dataset:

```
python setup_hybrid.py
```

Output should confirm: "MIGRATION COMPLETE" and "Admin Password set to: Admin123!"

3. **Run the Application:**

```
python app.py
```

Access the web interface at: `http://127.0.0.1:5000`

2.3 User Login Credentials

The system comes pre-seeded with role-specific accounts for testing:

Role	Username (Email)	Password	Access Rights
Admin	`admin@ltu.ac.uk`	`Admin123!`	User Management, Security Logs
Doctor	`dr.sterling@ltu.ac.uk`	`Doctor123!`	View/Edit Patients, Approve Appts
Patient	`patient9046@ltu.ac.uk`	`Patient123!`	View Own Profile, Book Appts

3. API Reference

Although the application renders server-side HTML, the internal routing acts as an API for the frontend logic.

3.1 Authentication Endpoints

Method	Endpoint	Access	Description
`POST`	`/login`	Public	Authenticates user against SQLite. Rate limit: 10/min.
`POST`	`/register`	Public	Creates SQLite auth record and encrypted MongoDB profile.

Method	Endpoint	Access	Description
GET	/logout	Authenticated	Clears session and logs audit event.

3.2 Patient Endpoints

Method	Endpoint	Access	Description
GET	/patient-dashboard	Patient	Renders the dashboard with decrypted PII and Risk Score.
POST	/update_profile	Patient/Doctor	Updates medical metrics (BMI, Glucose). Triggers Audit Log.
POST	/book_appointment	Patient	Creates appointment request if Risk Score > 30%.
GET	/download_report	Patient	Generates dynamic PDF with health persona and advice.

3.3 Doctor/Admin Endpoints

Method	Endpoint	Access	Description
GET	/doctor-dashboard	Doctor	Lists all patients (Search/Sort enabled) and appointment requests.
GET	/edit_patient/<id>	Doctor	Renders edit form for a specific patient ID.
GET	/process_appointment/<id>/<action>	Doctor	Updates appointment status (Approve/Neglect).
GET	/admin-dashboard	Admin	Shows system stats and full Security Audit Log.
POST	/admin/add_doctor	Admin	Creates new Doctor account.
GET	/admin/delete_user/<id>	Admin	Hard deletes user from **both** SQLite and MongoDB.

4. Security Controls Matrix

Vulnerability Class	Mitigation Strategy	Implementation File
Injection (SQLi)	Parameterized Queries (?)	app.py (Line 150)
Injection (NoSQLi)	Object Mapping via PyMongo	app.py (Line 160)
XSS (Cross-Site Scripting)	Input Sanitization (Bleach)	app.py (clean_input)
Sensitive Data Exposure	AES Encryption (Fernet)	app.py (encrypt_data)
Broken Access Control	Role checks (session['role'])	All Routes

Vulnerability Class	Mitigation Strategy	Implementation File
CSRF	Flask-WTF Tokens	HTML Forms
Brute Force	IP-based Rate Limiting	`app.py` (Line 20)