1. Project Summary

MedPulse is a secure, cloud-based platform designed to centralize and streamline patient medical records for hospitals, doctors, and patients in Eastern Africa. In a region where many health facilities still rely on paper-based records, MedPulse addresses inefficiencies, delays, and fragmented care caused by lost or inaccessible patient information.

The system comprises three interconnected portals: Hospital Portal, Doctor Portal, and Patient Portal. Hospitals can manage staff accounts, departments, and patient registrations. Doctors can access and update medical records, track consultations, prescriptions, and lab results. Patients can securely view their records, schedule appointments, and share information across facilities using a unique QR-coded ID, ensuring continuity of care.

Technically, MedPulse employs a RESTful API architecture with FastAPI, a PostgreSQL database using SQLAlchemy ORM, and a responsive frontend with HTML, CSS, and JavaScript. Security is ensured through JWT-based authentication, password hashing, and access controls. Additional features like PDF generation and QR code integration enhance usability.

Development followed an Agile methodology, emphasizing iterative testing, continuous feedback, and collaborative improvement. Challenges such as backend–frontend communication, database synchronization, and secure authentication were overcome with middleware, ORM relationships, transaction management, and rigorous testing.

MedPulse improves efficiency, data security, and continuity of care, enabling healthcare providers to make informed decisions while empowering patients with control over their health information. The project demonstrates a scalable, user-centered solution to modernize healthcare record management in resource-limited settings.

2. Project Journey

The MedPulse project evolved through a structured, iterative process guided by the Agile methodology, emphasizing collaboration, feedback, and continuous improvement.

1. Project Inception:
The team identified the critical challenge of fragmented paper-based medical records in Eastern Africa. Initial research into existing EMRs highlighted gaps in accessibility, offline use, and ease of adoption, leading to the definition of project objectives, scope, and SMART goals.

2. Planning and Requirement Gathering:
Team discussions and stakeholder analysis informed detailed user stories for hospitals, doctors, and patients. Functional and non-functional requirements were clarified, focusing on usability, security, and cross-hospital record sharing.

3. Development Iterations:
Tasks were divided among team members based on expertise. The backend and database were implemented first, revealing early challenges such as data synchronization issues and complex relational mappings. The frontend was developed alongside, with attention to user experience and intuitive workflows.

4. Collaboration and Feedback:
Weekly meetings, paired programming sessions, and internal demos ensured alignment. Feedback loops allowed quick identification and resolution of issues, including API route mismatches, CORS restrictions, and authentication errors.

5. Testing and Refinement:
Unit, integration, and end-to-end testing highlighted bugs and workflow gaps. The team iteratively refined the system, ensuring smooth patient record access, secure QR code functionality, and robust authentication.

6. Lessons Learned:
The team gained valuable experience in database management, backend-frontend integration, and collaborative software development. Key lessons included the importance of early database planning, clear communication between team members, and the value of iterative testing in improving system reliability.

3. Codebase

https://github.com/BodeMurairi2/medpulse-v2.git

4. Team Roles & Contributions

Faith IRAKOZE:

- Researched and compiled the background, problem statement, and objectives of the project.
- Defined the project scope, significance, and ethical considerations.
- Ensured that the project's introduction clearly communicated the real-world problem and the proposed solution.
- Designed the patient portal backend logic.

Laura KARANGWA KWIZERA:

- Conducted a detailed review of existing electronic medical record systems and related literature.
- Analyzed challenges faced by current EMRs in African healthcare contexts.
- Synthesized findings to justify the need for a localized, scalable solution like MedPulse.
- Designed the hospital portal frontend.

Bode MURAIRI:

- Designed the system architecture and development plan, including service layers, APIs, and deployment strategy.
- Created UML diagrams and workflow designs to guide development.
- Defined the Agile-based methodology to structure iterative development and testing.
- Designed the doctor portal backend logic

Pascal Louis NSIGO:

- Developed the relational database schema and entity relationships to ensure data integrity.

- Designed UML class diagrams to represent system entities and relationships.

- Collaborated closely with backend developers to implement ORM models and maintain consistency.
- Designed the hospital portal backend and all authentication logic

Vanessa UMWARI:

- Ensured that project demonstrations were clear, engaging, and visually aligned with technical content.

- Designed the patient portal frontend.

Maurice NSHIMYUMUKIZA:

- Produced the project presentations, including slide layout, design, and visual content.
- Supported team members by translating technical work into audience-friendly formats.
- Designed the doctor portal frontend.

Collaborative Contributions:

All team members participated in weekly meetings, brainstorming sessions, and code reviews.

Collective problem-solving was applied to address backend-frontend integration issues, testing challenges, and deployment decisions.

Shared documentation and version control practices via GitHub ensured transparency and traceability of contributions.

5. System Architecture

The MedPulse system is designed using a modular, RESTful architecture to ensure scalability, maintainability, and high performance. The architecture separates concerns into distinct layers, enabling clear division of responsibilities between components and smooth interaction between hospitals, doctors, and patients.

1. Users:

- **Hospitals:** Administer doctors, manage patient records, and generate reports.
- **Doctors:** Access patient records, create and update medical entries, and schedule consultations.
- **Patients:** View personal medical history, appointments, prescriptions, and receive notifications.

2. Frontend (UI Layer):

- Provides interactive dashboards tailored to each user role.
- Handles requests to backend APIs and presents data intuitively.
- Built with HTML, CSS, and JavaScript, with future plans to expand into React or React Native for mobile.

3. Backend (Service Layer):

- Implements core business logic, processes requests, and validates data.
- Handles authentication, authorization, and QR code generation.
- Built with **FastAPI**, supporting asynchronous processing for high concurrency.

4. API Layer:

- Exposes RESTful endpoints for communication between frontend and backend.
- Each resource (patients, doctors, hospitals) has clearly defined endpoints with input/output schemas.
- Ensures secure, role-based access to sensitive medical data.

5. Data Layer:

- **PostgreSQL database** stores structured medical records, prescriptions, appointments, and audit logs.
- **SQLAlchemy ORM** ensures integrity of relationships between entities and facilitates database migrations with Alembic.
- File storage (PDFs, images) is managed via **Cloudflare CDN**, allowing fast and reliable access.

6. External Services:

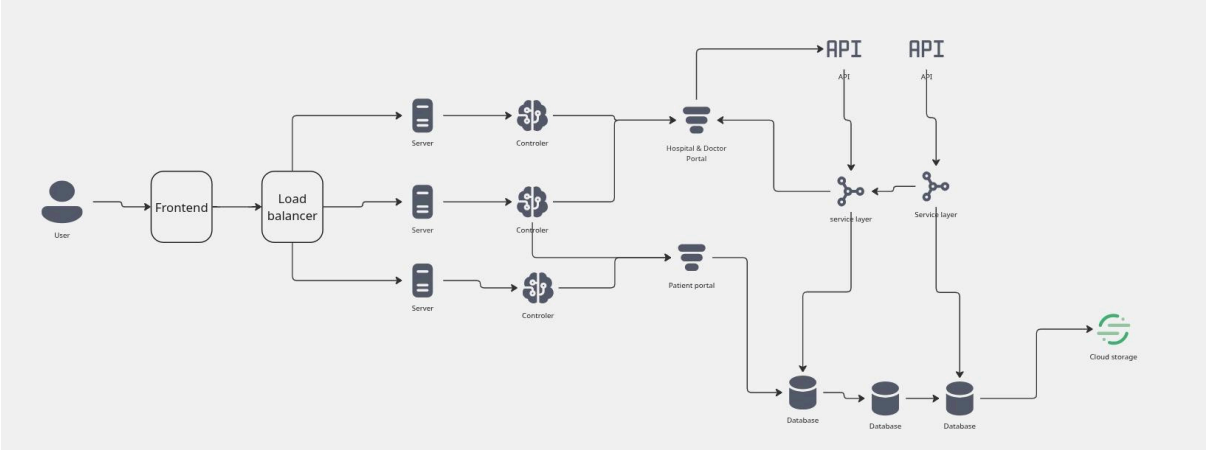
- Email notifications for password setup and appointment reminders.
- QR code generation for cross-hospital patient record access.

7. Infrastructure:

- The application server runs on **Uvicorn** with **Nginx** as a reverse proxy and static file server.
- Load balancing ensures high availability and evenly distributed traffic.
- Monitoring and logging are implemented via Prometheus and Grafana for system performance tracking.

Architecture Highlights:

- **Modularity:** Each layer can evolve independently without affecting others.
- **Security:** JWT-based authentication, role-based access, and HTTPS ensure patient data protection.
- **Scalability:** The system can handle multiple concurrent requests and can expand to additional hospitals easily.
- **Reliability:** Transaction management, database integrity, and rigorous testing ensure consistent and accurate record handling.

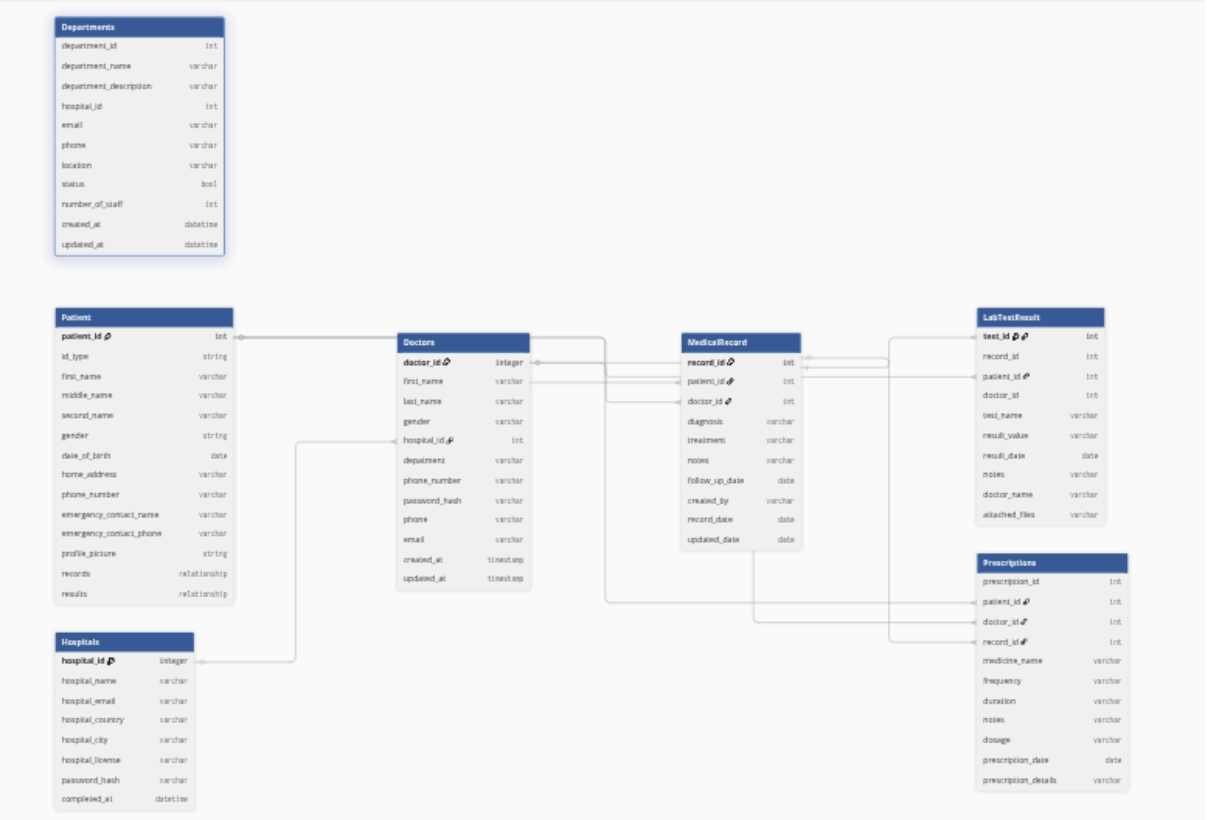


6. Technical Specifications

Component	Technology/Tool	Purpose
Backend Framework	FastAPI	Build RESTful APIs & handle requests
Server	Uvicorn (ASGI)	Run FastAPI app efficiently
Database	PostgreSQL	Store structured medical records
Authentication	JWT + Passlib (Argon2/Bcrypt)	Secure login & password hashing
PDF Generation	pdfkit	Export medical records as PDF
QR Codes	qrcode (Python) + Pillow	Generate shareable patient QR codes
Frontend	HTML, CSS, JavaScript	User interface for patients, doctors, hospitals
Testing	Unittest, Pytest, Jest, Postman, request (python), Curl	Unit, integration, and end-to-end testing
Version Control	Git + GitHub	Collaboration, tracking changes, code review

Technical Diagrams

1. ERD Diagram



Technical Challenges

1. Backend–Frontend Communication Issues

One of the major technical challenges we faced was establishing a smooth communication flow between the frontend and the backend.

The frontend struggled to correctly send requests and process responses due to:

- Mismatched API routes
- Incorrect request payload formats
- CORS (Cross-Origin Resource Sharing) restrictions
- Authentication headers are not being passed correctly

How We Overcame It

- Implemented CORS middleware in FastAPI to allow the frontend domain to communicate with the backend.
- Ensured all API routes followed a clean, consistent structure (e.g., **/api/v1/patients**).
- Created shared API documentation automatically using FastAPI's built-in Swagger UI, helping the frontend team understand the required input/output formats.
- Standardized all request/response bodies using Pydantic schemas, which prevented inconsistent data being passed between layers.

2. Data Synchronization Across Multiple Tables

Our database relies on relationships between patients, doctors, hospitals, prescriptions, and medical records.
Challenges included:

- Ensuring foreign keys were correctly created
- Managing updates that affected related tables
- Avoiding race conditions during asynchronous operations

How We Overcame It

- Used SQLAlchemy ORM relationships to maintain database integrity.
- Implemented transaction management to ensure that grouped operations either all succeeded or failed together.
- Wrote tests using *pytest* and *pytest-asyncio* to verify that related tables synchronized correctly.

3. API Testing and Debugging

During development, some endpoints failed due to schema mismatches or missing fields.

How We Overcame It

- Used FastAPI's interactive docs to manually test routes.
- Added strict Pydantic validators to catch errors early.
- Logged detailed error messages using FastAPI's exception handling and Python's logging module.

4. Database Seeding Errors

While preparing test data, we encountered:

- Missing fields in seed objects
- Renamed columns (e.g., using **date_issued** instead of **prescription_date**)
- Missing primary keys (**id** vs **patient_id**)

How We Overcame It

- Reviewed all ORM model definitions to ensure correct field names.
- Updated the seeding script to match the updated database structure.
- Reran Alembic migrations to ensure schema consistency.

User-facing Material

The **MedPulse** project is a web application designed to simplify the keeping, accessing, updating, and sharing of medical records among doctors in hospitals and patients. Specifically, **MedPulse** will enable doctors to immediately have access to the medical records of the patients they are treating, which can help ease their treatment process. Let's navigate through all three portals of **MedPulse**:

1. Hospital Portal

- We will be working with hospitals that have expressed interest in our MedPulse, and then we will give them access to create their account.

- After creating their account, they will be directed to the dashboard, where different features are located.

- Hospitals will be the only authorized entities to create their doctors' accounts and register patients, assigning them unique IDs to use across hospitals.

- Through this portal, the hospital administrators will be able to edit and update their services and the staff accounts they hold.

2. Doctor Portal

- After the hospital has created their staff(doctors) accounts using their email, a default password will be generated, and they will be given authorization to change it to one they prefer.

- The Doctors will be required to log in to access and interact with their dashboard and access the medical records of their patients.

- If a doctor had a consultation session with a patient or took a laboratory test, they would be able to fill in the details through our app, which replaces the usual paper form.

- And as the patient will also have an account, the doctor will easily access the patient's medical records that will have been added to the patient's portal by other doctors. This information will be taken into consideration by the doctor in treatment and prescriptions.

3. Patient portal

- Similar to the doctor, hospitals will create a patient's account using the patient's email, and a password will be automatically generated. The email containing the patient's credentials will be sent through email, permitting them to change their password.

- To access any other thing, the patient must log in first and be able to navigate through the portal.

- For the medical records of each patient, access is set to 'only view' to prevent the patients from adding irrelevant information that can affect their next treatment with a different doctor.

- Another feature on the patient's portal is a QR Code, which the doctor will scan to get any information related to the current situation or treatment.

- The ID provided by the hospital during the registration of a new patient is not only unique for patients at that specific hospital, but also for all the hospitals that the patient visits, allowing cross-sharing of information among hospitals.

Briefly, here is the diagram that shows it: