

COMP3335

Database Security

Group Project Report (Group 1)

MedTest Lab

Group Members:

Bora Aktas 24010776x

Buse Oznur Ozkan 24010868x

Azan AKBAR 24032229d

Severin Alexander HEITZER 24011011X

Shomique HAYAT 24010822X

Table Of Contents:

Introduction	3
System Architecture	4
Database Design	6
Security Implementation	17
Usage and Demonstration	20
Conclusion	30

Introduction

This project focuses on designing and securing the database of a medical laboratory, MedTest Lab, while ensuring compliance with medical data standards. By leveraging modern technologies like Docker, Transparent Data Encryption (TDE), and role-based access control (RBAC), our system ensures secure storage and processing of sensitive medical data. Additionally, it includes real-time monitoring and alerting to detect and respond to potential security threats.

We focused on it to be user friendly and efficient in managing work loads for the different roles we have had to make functionalities for while ensuring resilience and security in the design of it all.

All in all, our project is a comprehensive implementation of a secure database system for MedTest Lab. It integrates modern technologies such as Docker, Transparent Data Encryption (TDE), and real-time monitoring with Prometheus and Grafana. By using a Flask-based alert system and PHP for the user interface, the system demonstrates advanced techniques for ensuring medical data security.

Project Objectives:

- Design and secure a database for MedTest Lab.
- Comply with medical data standards through encryption and role-based access control (RBAC).
- Develop a secure web interface to interact with the database.
- Monitor the database for suspicious queries or potential SQL injection attempts.

Scope of Work:

- Utilize a Dockerized LAMP stack.
- Implement Transparent Data Encryption (TDE) with Percona.
- Integrate monitoring tools (Prometheus, Grafana, and Loki) for database activity tracking.
- Provide a user-friendly and secure front-end in PHP.

System Architecture

The system architecture for our project is a modular and secure design built around Dockerized microservices, ensuring scalability, maintainability, and security. At its core is a **Percona database server** with Transparent Data Encryption (TDE) to safeguard sensitive medical data.

The **web interface**, implemented using PHP 8.3 with Apache, acts as the user interaction layer, providing secure access to different functionalities for patients, lab staff, and secretaries.

A robust **monitoring stack** integrates Prometheus, Grafana, and Loki to track database activity, visualize metrics, and generate real-time alerts for suspicious actions. Logs are aggregated with Promtail and stored in Loki for detailed auditing.

The architecture also features **Vault**, a key management system to securely store and manage encryption keys. These components are seamlessly orchestrated with Docker Compose, ensuring an isolated, consistent, and efficient deployment environment.

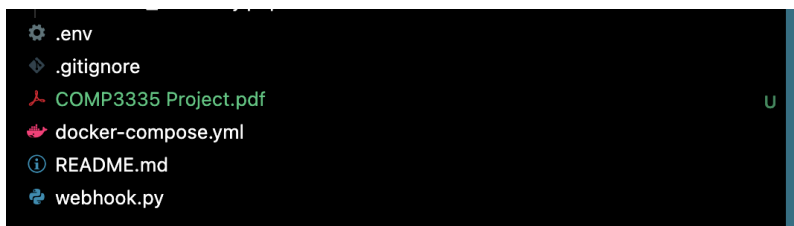
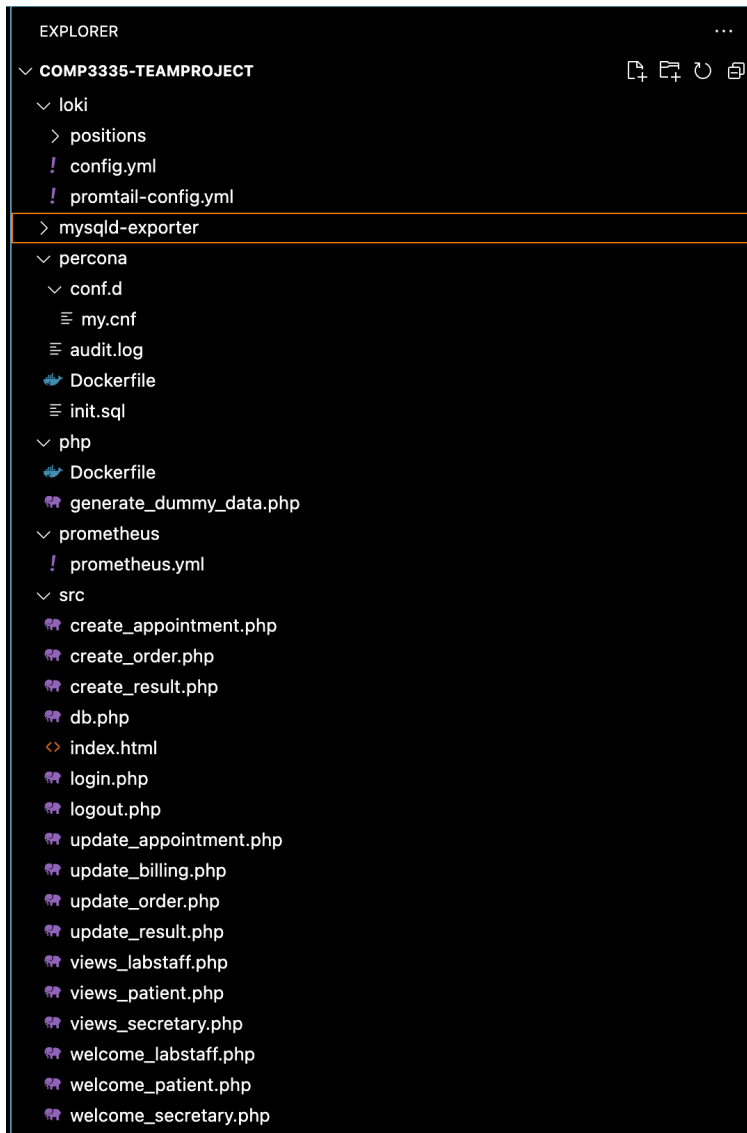
Our system is built on a Dockerized LAMP stack with the following components:

Components:

- **Web Server:**
 - PHP 8.3 with Apache (LAMP stack) for user interaction and secure data handling.
- **Database Server:**
 - Percona 8.0 with Transparent Data Encryption (TDE).
 - Configured for role-based access and encryption.
- **Monitoring Stack Tools:**
 - **Prometheus:** Collects and aggregates metrics.
 - **Grafana:** Real-time monitoring and alerts trigger dashboard.
 - **Loki & Promtail:** Log aggregation for suspicious query tracking.
- **Vault:**
 - Manages and secures encryption keys.
- **Alert System:**
 - Flask-based webhook to capture and display alerts.

Dockerized Setup:

- Fully containerized services for seamless deployment and consistent environment configuration.
- Contains components for secure database operation and monitoring.



Database Design

The database design of our project follows a comprehensive schema tailored to meet the requirements of MedTest Lab. It is structured to handle sensitive medical and financial data securely while supporting role-based access for Lab Staff, Secretaries, and Patients.

The schema includes tables for managing user credentials, roles, patients, staff, tests, orders, appointments, results, and billing. To ensure data integrity and security, we utilize foreign key constraints, triggers for status updates, and stored procedures for consistent operations.

Sensitive information is encrypted using AES-256-CBC, and Transparent Data Encryption (TDE) is enabled at the database level. Additionally, predefined views simplify role-specific data access, and role-based privileges enforce the principle of least privilege, ensuring users only access the data necessary for their roles.

Database Schema

Users and Roles

- **Users Table:**
 - **Columns:**
 - `userID` (Primary Key)
 - `email` (Unique)
 - `password` (Hashed)
 - **Description:** Stores login credentials for all users.
- **Roles Table:**
 - **Columns:**
 - `roleID` (Primary Key)
 - `roleName` (e.g., labStaff, secretary, patient)
 - **Description:** Defines roles for users.
- **UserRoles Table:**
 - **Columns:**
 - `userID` (Foreign Key referencing Users)
 - `roleID` (Foreign Key referencing Roles)
 - **Description:** Maps users to their roles.

Patients

- **Columns:**
 - `patientID` (Primary Key, Foreign Key referencing Users)
 - `patientSSN` (Unique, Encrypted)
 - `firstName`, `lastName`, `birthDate`, `phoneNo`, `insuranceType`

- **Description:** Stores patient demographic and insurance information.

Staff

- **Columns:**
 - `staffID` (Primary Key, Foreign Key referencing Users)
 - `firstName`, `lastName`, `phoneNo`, `staffRole` (e.g., labStaff, secretary)
- **Description:** Stores details about lab staff and secretaries.

Test Catalog

- **Columns:**
 - `testID` (Primary Key)
 - `testCode` (Unique)
 - `testName` (Unique)
 - `testCost`, `testDescription`
- **Description:** Contains information about available tests.

Orders

- **Columns:**
 - `orderID` (Primary Key)
 - `patientID` (Foreign Key referencing Patients)
 - `labStaffOrderID` (Foreign Key referencing Staff)
 - `testID` (Foreign Key referencing TestCatalog)
 - `orderDate`
 - `orderStatus` (e.g., Pending Appointment, Pending Result, Completed)
- **Description:** Logs test orders for patients and tracks their status.

Appointments

- **Columns:**
 - `appointmentID` (Primary Key)
 - `orderID` (Foreign Key referencing Orders)
 - `secretaryID` (Foreign Key referencing Staff)
 - `appointmentDateTime`
- **Description:** Schedules appointments for patients and links them to test orders.

Results

- **Columns:**
 - `resultID` (Primary Key)
 - `orderID` (Foreign Key referencing Orders)
 - `labStaffResultID` (Foreign Key referencing Staff)
 - `interpretation`, `reportURL` (Encrypted)

- **Description:** Stores test results along with the responsible lab staff.

Billing

- **Columns:**
 - **billingID** (Primary Key)
 - **orderId** (Foreign Key referencing Orders)
 - **billedAmount** (Encrypted)
 - **paymentStatus, insuranceClaimStatus**
- **Description:** Tracks billing information for test orders.

This screenshot shows the environment variable configuration for the database system in the `.env` file, which includes user credentials, role-based encryption keys (e.g., `ENCRYPTION_KEY`), and the encryption method (`AES-256-CBC`). These settings ensure secure and separate access for each role (Admin, Patient, Lab Staff, Secretary) while supporting Transparent Data Encryption (TDE) for protecting sensitive data. Also, for the encryption keys for the encrypt tables, we use the HashiCorp vault and save its credentials in the docker-secrets to encrypt data at rest and save encryption keys securely. You can see it from our docker-compose.yml file.


```

services:
  percona:
    image: percona:8.0.36-28
    platform: linux/amd64
    container_name: COMP3335_db
    build: ./percona
    env_file:
      - .env
    volumes:
      - ./percona/init.sql:/docker-entrypoint-initdb.d/init.sql
      - mysql-data:/var/lib/mysql
    secrets:
      - source: keyring_vault_conf
        target: /run/secrets/keyring_vault.conf
        mode: 0440 # read-only access
    ports:
      - "3306:3306"
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "percona"]
      interval: 10s
      timeout: 5s
      retries: 3
    command: >
      --plugin-load=audit_log=audit_log.so
      --default-table-encryption=ON
      --early-plugin-load=keyring_vault=keyring_vault.so
      --loose-keyring-vault-config=/run/secrets/keyring_vault.conf
    depends_on:
      vault:
        condition: service_healthy

  vault:
    image: vault:1.13.3
    container_name: COMP3335_vault
    restart: on-failure:10
    ports:
      - "8200:8200"
    environment:
      VAULT_ADDR: 'http://127.0.0.1:8200'
      VAULT_DEV_ROOT_TOKEN_ID: '00000000-0000-0000-0000-000000000000'
    cap_add:
      - IPC_LOCK
    command: server -dev -dev-root-token-id="00000000-0000-0000-0000-000000000000"
    healthcheck:
      test: ["CMD", "vault", "status"]
      interval: 10s
      timeout: 5s
      retries: 3

```

Key Segments of the Database Design

Procedures

- **Purpose:** Procedures automate database operations, ensuring consistency and reducing the likelihood of errors during user creation.
- **Key Procedures:**
 - **insertPatient:**
 - Inserts a new patient into the database.
 - Maps the patient to the **patient** role by assigning the appropriate **roleID**.
 - Ensures consistent insertion across multiple related tables (**users**, **userRoles**, **patients**).
 - **insertStaff:**

- Similar to `insertPatient`, but for lab staff and secretaries.
- Assigns specific roles (`labStaff` or `secretary`) and links them to the corresponding staff table.

Triggers

- **Purpose:** Automate status updates and enforce rules at specific events (e.g., when a row is inserted or deleted).
- **Key Triggers:**
 - **`updateOrderStatusCreateAppointment`:**
 - Automatically sets the `orderStatus` to `Pending Result` when an appointment is created.
 - **`updateOrderStatusDeleteAppointment`:**
 - Reverts the `orderStatus` to `Pending Appointment` when an appointment is deleted.
 - **`checkOrderStatusBeforeInsertTestResult`:**
 - Validates the current `orderStatus` before allowing test results to be added, ensuring logical consistency.

Role-Based Access Control

- **Purpose:** Enforce the principle of least privilege by assigning different permissions to Patients, Lab Staff, and Secretaries.
- **Role Privileges:**
 - **Patients:**
 - Can view their orders, results, and billing via predefined views (`patientOrders`, `patientResults`, `patientBillings`).
 - **Lab Staff:**
 - Can manage orders and results, including creating, updating, and deleting them.
 - Access patient information and test catalogs to handle lab operations.
 - **Secretaries:**
 - Manage appointments and billing information.
 - Access and update order details to coordinate lab activities.

Views

- **Purpose:** Simplify data access for specific roles while hiding unnecessary details.
- **Key Views:**

- **patientOrders:**
 - Displays orders linked to a specific patient, including test names and lab staff details.
- **labStaffOrders:**
 - Shows orders assigned to a particular lab staff member, along with patient details and statuses.
- **secretaryAppointments:**
 - Provides a schedule of appointments managed by a specific secretary.

Tables

- **Users and Roles:**
 - Centralized tables for user authentication (**users**) and role definitions (**roles**), linked via **userRoles**.
- **Domain-Specific Tables:**
 - **patients:** Stores sensitive patient data, including encrypted SSNs and contact details.
 - **staffs:** Captures staff details with their specific roles (**labStaff** or **secretary**).
 - **orders:** Tracks test orders with their status, linked to patients, tests, and lab staff.
 - **appointments:** Links orders with secretaries for scheduling.
 - **results:** Stores test results, including report URLs and interpretations.
 - **billings:** Manages financial data such as billed amounts and insurance claims.

Security Implementation

The security implementation for our project focuses on robust mechanisms to protect sensitive data, monitor activities, and identify potential threats in real-time. Below are the key aspects of our security measures:

Our project employs a multi-layered security architecture to ensure the protection of sensitive medical data and mitigate risks of unauthorized access. Central to this is Transparent Data Encryption (TDE) enabled in Percona, which encrypts data at rest using AES-256-CBC encryption. The Role-based access control (RBAC) enforces strict data access policies, ensuring users (patients, lab staff, and secretaries) only interact with data relevant to their roles.

A Grafana-based monitoring dashboard integrates with Prometheus and Loki to track database activities and highlight anomalies such as SQL injection attempts or unauthorized sensitive table access. The alerting system, built using Flask, provides real-time notifications for suspicious activities, leveraging custom dashboards for threat analysis. Additionally, SQL triggers ensure consistency in database operations, while stored procedures and encryption keys safeguard sensitive operations and data transmissions.

Key Features and Functionalities:

Transparent Data Encryption (TDE):

- AES-256-CBC encryption secures patient, staff, and billing data at rest.
- Managed through encryption keys specified in `keyring_vault.conf`.

Role-Based Access Control (RBAC):

- Database roles (`patient`, `labStaff`, `secretary`) grant limited privileges for accessing and modifying specific tables.
- Privileges defined and enforced at the MySQL database level.

Monitoring and Alerting:

- **Prometheus:** Collects metrics for real-time database activity.
- **Loki:** Aggregates logs, enabling detection of SQL injection attempts and unauthorized access patterns.
- **Grafana Dashboards:**
 - **MySQL Overview Dashboard:** Displays system health, queries per second, and active connections.
 - **Suspicious Query Monitoring:** Identifies SQL injection patterns and excessive query rates.

Webhook for Real-Time Alerts:

- `alert_page.py` Flask app captures alerts from Grafana and displays them on a custom dashboard.
- Alerts include attempts to access sensitive tables or execute unauthorized queries.

SQL Injection Prevention:

- Implemented through triggers and constraints that validate inputs and block risky operations.
- SQLMap tests in `simulate_attacks.py` evaluate system defenses.

Triggers and Procedures:

- **Triggers:**
 - Automatically update the status of orders and appointments.
 - Prevent inserting results for orders in invalid states.
- **Stored Procedures:**
 - Automate user creation and role assignment for patients and staff.

Dashboards and Metrics:

- **Sensitive Table Access Panel:** Tracks queries involving sensitive information.
- **SQL Injection Attempts Panel:** Highlights unusual query patterns indicative of injection.
- **Query Rate Panel:** Detects anomalies in query frequency.

Percona Security Features:




























- **Audit Logs:** Captures all user interactions with the database.
- **Key Management:** Enforced through the `keyring_vault.conf` file for TDE.

Simulated Attack Testing:

- `simulate_attacks.py`: Executes SQL injection and access control tests to verify the system's resilience.

Prometheus Configuration:

- Monitors database metrics and integrates with Grafana for visualization (`prometheus.yml`).

- ✓  **COMP3335-TeamProject-main** ~/Downloads/COMP3335-TeamP
 - >  .idea
 - ✓  grafana
 - ✓  dashboards
 -  MySQL Overview.json
 -  MySQL Suspicious Query Monitoring.json
 - ✓  provisioning
 - ✓  alerting
 -  alerting.yml
 -  contact_points.yml
 -  notification_policies.yml
 - ✓  dashboards
 -  dashboard.yml
 - ✓  datasources
 -  loki.yml
 -  prometheus.yml
 - ✓  loki
 - ✓  positions
 -  positions.yaml
 -  config.yml
 -  loki_dashboard.json
 -  promtail-config.yml
 - >  mysqld-exporter
 - ✓  percona
 -  Dockerfile
-  init.sql
-  keyring_vault.conf

Usage and Demonstration

The web application includes interactive dashboards and functionalities tailored for different roles—patients, lab staff, and secretaries—providing secure, efficient, and user-friendly interfaces for medical test management. Each dashboard is specifically designed to meet the operational needs of its respective user role, with clear workflows and detailed information display.

Project Setup Instructions

1. **Prerequisites:**
 - **Install Docker, Python, and Pip.**
 - **Ensure your user has permissions to execute Docker commands.**
2. **Navigate to the project folder.**
3. **Install Python dependencies:**
 - `pip install -r requirements.txt`
4. **Build and start Docker containers:**
 - `docker-compose up -d --build`
5. **Start the alert system:**
 - `python alert_page.py`
6. **Simulate attacks:**
 - `python simulate_attack.py`

Access the System Components:

- **Website:** <http://localhost/>
- **phpMyAdmin:** <http://localhost:8081/>
- **Grafana:** <http://localhost:3000/dashboards> (username: `admin`, password: `admin`)
- **Alert System:** <http://localhost:3200/alerts>

Patient Dashboard

1. **Functionalities:**
 - **View Test Orders:** Patients can view all their test orders, along with the status, such as "Pending Appointment," "Pending Result," or "Completed."
 - **View Bills:** Patients can review their billed amounts, insurance claim statuses, and payment statuses.
 - **Navigation:** The dashboard is equipped with a clear and intuitive dropdown menu for accessing different functionalities.
2. **Demonstrated Features:**

- Displays patient-specific information for viewing orders and bills, ensuring patients are aware of pending tasks like payments or appointments.

Lab Staff Dashboard

1. Functionalities:

- **Create and Update Orders:** Lab staff can create new test orders for patients, selecting tests and specifying order dates.
- **View and Update Results:** They can upload and update test results, including report URLs and medical interpretations.
- **Navigation:** Accessible menus for switching between orders and results.

2. Demonstrated Features:

- Showcases how lab staff interact with orders and results for seamless test management, ensuring accuracy in test reporting.

Secretary Dashboard

1. Functionalities:

- **Manage Appointments:** Secretaries can create, update, and delete appointments for pending orders, ensuring the scheduling of sample collection and other procedures.
- **Update Billing and Payments:** Handles payment statuses and ensures proper billing updates for insurance claims.
- **Navigation:** Streamlined options for managing appointments and financial details.

2. Demonstrated Features:

- Illustrates the secretary's role in ensuring smooth coordination between patients and lab staff.

General Demonstrations

1. Order and Billing Management:

- **Payment Status Update:** Secretaries can update payment statuses to reflect whether bills are paid or pending.
- **Order Creation:** Lab staff can efficiently create new orders for patients, choosing the required test and specifying the order details.
- **Billing Overview:** Patients and secretaries have access to comprehensive billing details, streamlining financial management.
- Demonstrated by images of "Create a New Order" and "Update Payment Status."

2. Appointment Scheduling:

- Demonstrated with images of creating and updating appointments, showing how secretaries manage schedules efficiently.

3. Test Results Management:

- Features include uploading results with URLs and adding interpretations, demonstrated in the "Create Result" image.

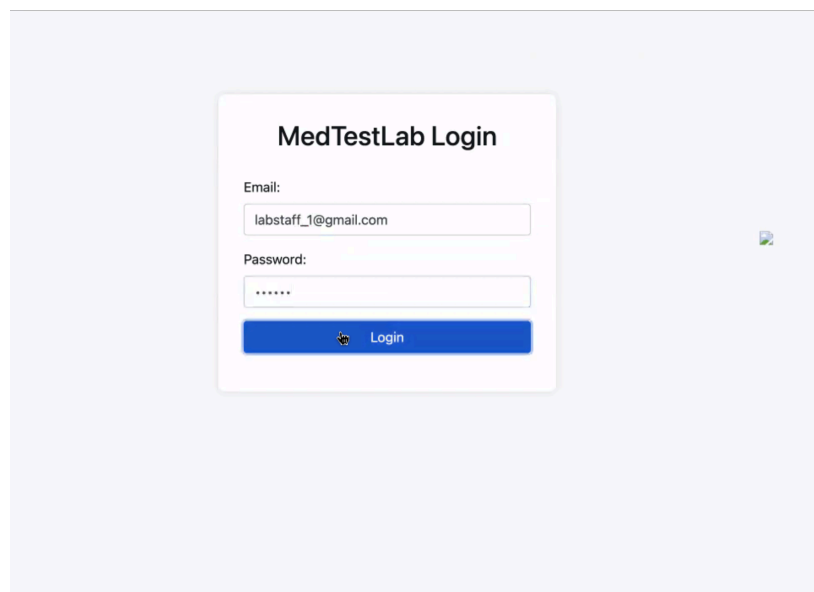
Navigation and User Experience

Each dashboard provides role-specific functionality while ensuring a seamless user experience:

- **Role-Based Access:** Ensures patients, lab staff, and secretaries only access relevant information.
- **Dynamic Menus:** Options displayed dynamically based on user roles to simplify navigation.
- **Visual Clarity:** Clean interfaces with clear call-to-action buttons (e.g., "Create Order," "Update Billing").

These screenshots collectively demonstrate the efficient workflow management provided by the system, ensuring secure, user-friendly, and task-specific interfaces.

- Welcome Login Screen



- Welcome Splash Screen for Lab Staff

Welcome, LabStaff_1_firstName LabStaff_1_lastName

MedTestLab Lab Staff Page

Task:

View Orders

Execute

Logout

- Welcome Splash Screen for Patients

Welcome, patient_1_firstName patient_1_lastName

MedTestLab Patient Page

Task:

View My Orders

Execute

Logout

- Welcome Splash Screen for Secretaries

Welcome, Secretary_1_firstName Secretary_1_lastName

MedTestLab Secretary Page

Task:

View Appointments

Execute

Logout

- Creating a new appointment in the role of a Secretary

Create a New Appointment

Select Order:

Order ID: 2

Appointment Date and Time:

19/12/2024 04:42 pm

Create Appointment

Back to Welcome Page

- A view page for secretaries with key information such as booked appointments and patient information

Appointments

Order	Appointment Date & Time
1	2020-11-10 10:00:00

Back

Create Appointment

Update Appointment

Logout

- A Patient can view their bills, insurance claim status and payment status

Your Bills

Order	Test Name	Billed Amount	Insurance Claim Status	Payment Status
1	Blood Test	90	Claimed	Not Paid
2	Urine Test	45	Claimed	Not Paid
5	Ultrasound	270	Claimed	Not Paid

Back

Logout

- Patients can view the tests and their statuses

Your Test Orders

Order	Test Name	Physician	Order Date	Appointment Date & Time	Order Status
1	Blood Test	LabStaff_1_firstName LabStaff_1_lastName	2020-11-01	2020-11-10 10:00:00	Completed
2	Urine Test	LabStaff_2_firstName LabStaff_2_lastName	2020-11-02	N/A	Pending Appointment
5	Ultrasound	LabStaff_1_firstName LabStaff_1_lastName	2024-12-20	N/A	Pending Appointment

[Back](#)[Logout](#)

- Creating a new Order in the role as a Lab Staff Member

Create a New Order

Select Patient:

patient_1_firstName patient_1_lastName (123-45-6789) ▾

Select Test:

Blood Test ▾

Order Date:

dd/mm/yyyy 📅

Create Order

Back to Welcome Page

- A Lab Staff can create and log a new result

Create a New Result

Select Order:

Report URL:

https://www.example.com/report1

Interpretation:

Create Result

Back

Logout

- A Lab Staff having a view of the test results including a report URL and a respective medical interpretation for each

Test Results

Order	Test Name	Patient Name Surname	Physician Name Surname	Pathologist Name Surname	Report	Interpretation
1	Blood Test	patient_1_firstName patient_1_lastName	LabStaff_1_firstName LabStaff_1_lastName	LabStaff_2_firstName LabStaff_2_lastName	View Report	Normal

Back

Create Result

Update Result

Logout

- A Lab Staff wanting to update an existing order must select an existing order they want to modify from the system.

Update an Existing Order

Select Order:

Order ID: 4

Select Order

Back

- A Lab Staff having a view to update an existing order they have already created or is already in the system

Update an Existing Order

Select Patient:

patient_3_firstName patient_3_lastName (543-21-9876)

Select Test:

MRI

Order Date:

04/11/2020

Update Order

Delete Order

Back

- A Secretary having a view to see all billings with the appropriate Patient SSN and the statuses of the payment and the insurance claim. These are logs for the role of a secretary

Billings

Order	Billed Amount	Insurance Claim Status	Payment Status
1	90	Approved	Pending

Back

Update Billing

Logout

- A Secretary has the ability to change the payment status of an order that is on the system

Change Payment Status of an Order

Select Order:

Order ID: 1

Payment Status:

Paid

Change Payment Status

Back to Welcome Page

Conclusion

In this project, we successfully developed and implemented a secure, efficient, and user-friendly database system for MedTest Lab. By leveraging modern technologies such as Docker, Percona's Transparent Data Encryption (TDE), and role-based access control (RBAC), the system ensures the protection of sensitive medical data while maintaining high usability across multiple user roles. The integration of monitoring and alerting tools like Prometheus, Loki, and Grafana provides real-time insights into database activity and safeguards against potential threats, such as SQL injection.

The modular design of the system supports scalability and maintainability, allowing MedTest Lab to manage patient records, test orders, appointments, results, and billing seamlessly. Comprehensive dashboards for patients, lab staff, and secretaries demonstrate the alignment of functionalities with role-specific needs. These dashboards offer secure and intuitive interfaces, enhancing operational efficiency and transparency.

By simulating attacks and monitoring database activity, the system has been rigorously tested for resilience, confirming the effectiveness of its security measures. The inclusion of encryption protocols, role-specific privileges, and automated triggers/procedures further strengthens the integrity and confidentiality of the data.

This report highlights the significant strides made toward achieving a robust and secure database architecture for medical laboratories, showcasing a blend of innovation, precision, and adherence to security standards. MedTest Lab's database system stands as a reliable solution for managing sensitive medical workflows while safeguarding patient information.