Implementation Of Secure LIMS With Encryption and Authentication

SURYA PRAMOD VADAPALLI, LAASYA VAJJALA[1]

[1] Comuter Science Department, Washington State University, Pullman, Washington, 99163
[*] s.vadapalli@wsu.edu, laasya.vajjala@wsu.edu

Compiled March 24, 2025

ABSTRACT

Laboratory Information Management Systems (LIMS) play a critical role in managing laboratory operations, including sample tracking, data storage, and reporting. However, many existing LIMS lack robust security features, such as native encryption and multi-factor authentication (MFA), leaving sensitive data vulnerable to breaches. In this project, we address these gaps by designing and implementing a secure LIMS-like system from scratch using modern technologies.

The developed system focuses on three key areas: user authentication, data security, and functionality. It incorporates MFA for enhanced user authentication, ensuring that access is protected by multiple layers of verification. Sensitive data fields, such as sample names and types, are encrypted using AES-256 encryption to safeguard data at rest, even if the database is compromised. Additionally, the system includes full Create, Read, Update, and Delete (CRUD) operations for managing sample records, seamlessly integrating encryption to maintain data security during these processes.

The project began with an attempt to adapt an open-source LIMS codebase built on Python 2.7, but compatibility issues led to the decision to develop a new system using Django and Python. The result is a simplified, yet secure, system that prioritizes usability and security. Features such as Bootstrap-based forms and audit logs enhance user experience and system transparency.

This work demonstrates how a combination of modern frameworks and best practices can build a LIMS that is both functional and secure. The system is suitable for small to medium laboratories and serves as a foundation for future enhancements, such as real-time monitoring and advanced analytics. This paper details the design, implementation, and evaluation of the system, offering insights into addressing security challenges in laboratory management.

http://dx.doi.org/10.1364/ao.XX.XXXXXX

1. INTRODUCTION

A. The Role of LIMS in Modern Laboratories

Laboratory Information Management Systems (LIMS) are essential tools in modern laboratories to manage data, samples, and workflows efficiently. They help laboratories maintain compliance, improve productivity, and ensure data traceability. As laboratories handle increasingly sensitive and valuable data, the need for secure systems to manage these data has grown significantly.

However, despite their importance, many existing LIMS systems fail to prioritize security. Sensitive data, such as sample information, test results, and patient details, can become vulnerable to breaches if not adequately protected. In our analysis, we observed that many LIMS lack features like data encryption or native multi-factor authentication (MFA), leaving critical gaps in their security frameworks. This highlights the urgent need for secure LIMS solutions.

B. Why is Security in LIMS Important?

Security in LIMS is vital for protecting sensitive data from unauthorized access, data breaches, and internal threats. Laboratories often deal with proprietary research, patient data, and confidential results, all of which need to be safeguarded to maintain trust and meet regulatory compliance.

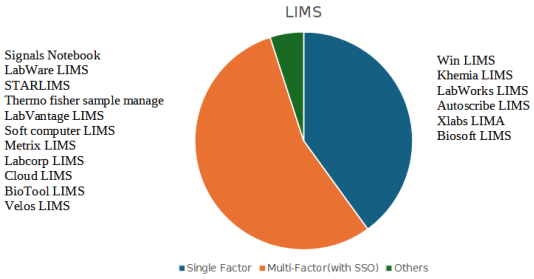

Fig. 1. Analysis of various LIMS

Our analysis revealed the following:

- 40 percent of LIMS systems rely solely on single-factor authentication, such as a username and password.
- 55 percent of systems integrate multi-factor authentication through third-party Single Sign-On (SSO) providers.

The remaining systems lack any structured security layers. This means that a significant portion of LIMS systems either fail to meet security standards or depend on external tools, which can introduce dependency risks. Furthermore, many LIMS do not encrypt sensitive data, exposing it to potential misuse in the event of database compromise.

C. How is Our System Different?

To address the shortcomings in existing LIMS, we developed a secure LIMS-like system from scratch. Our system is built to:

- Integrate Security as a Core Component: Unlike traditional LIMS, which often add security as an afterthought, our system incorporates features like encryption and MFA from the ground up.
- Eliminate Third-Party Dependencies: By implementing native MFA and encryption, we reduce reliance on third-party solutions, ensuring the system remains independent and customizable.
- Protect Sensitive Data at All Levels: The system encrypts critical fields (e.g., sample names and types) at the database level, ensuring data remains protected even if unauthorized access occurs.

D. Insights from Our Analysis

Our project was motivated by the gaps identified in current LIMS systems:

- Single-Factor Authentication Limitations: Password-based systems alone are no longer sufficient to protect sensitive data from sophisticated attacks.
- Dependency on External MFA Providers: While SSO systems offer convenience, they introduce risks if the third-party service is compromised or unavailable.
- Lack of Encryption: Without field-level encryption, sensitive data remains exposed in plaintext within databases.
- By addressing these gaps, we created a system that integrates both security and usability. It not only meets the functional requirements of a LIMS but also ensures that sensitive data is protected at every step.

E. Objectives of the Project

The primary objectives of our project were:

- To design and implement a secure LIMS-like system with built-in security features.
- To provide robust user authentication through native MFA.
- To ensure data security by encrypting sensitive fields.
- To deliver a user-friendly interface for managing laboratory data securely.

This paper explores how we achieved these objectives, from design to implementation, and evaluates the impact of our system on LIMS security.

2. PROBLEM DEFINITION AND RELATED WORK

A. Problem Definition

The primary challenge faced by laboratories today is the lack of robust security mechanisms in Laboratory Information Management Systems (LIMS). These systems are integral to managing samples, test results, and sensitive laboratory data, but they often fail to address key aspects of data security and user authentication.

Key Issues Identified:

Weak Authentication Mechanisms:

- Many LIMS systems still rely on single-factor authentication (username and password), which is vulnerable to credential theft and brute-force attacks.
- Multi-factor authentication (MFA), though available in some systems, is typically implemented via third-party Single Sign-On (SSO) providers, introducing dependency risks.

Lack of Data Encryption:

- Sensitive information, such as sample types, patient data, or proprietary research results, is often stored as plaintext in databases.
- This exposes critical data to risks such as database breaches, insider threats, and unauthorized access.

Limited Integration of Security into Core Functionality:

- Security features like encryption and authentication are frequently added as external modules rather than being integrated into the core design.
- This makes the systems harder to maintain and less reliable.

Minimal Focus on Accountability:

Many LIMS systems lack robust audit logging to track user actions, which is essential for accountability and compliance.

Dependence on Legacy Systems:

- Some laboratories continue to use outdated LIMS platforms built on legacy technologies like Python 2.7, which are incompatible with modern libraries and tools.

Impact of These Issues: The absence of integrated security measures leaves laboratory data vulnerable to breaches, compromises regulatory compliance, and reduces trust in the system. A secure, modern LIMS is needed to bridge these gaps and ensure data confidentiality, integrity, and availability.

Related Work

In recent years, various attempts have been made to enhance the security of LIMS systems. However, these solutions often focus on specific aspects, such as user authentication or data encryption, without addressing security comprehensively. Below are key findings from existing literature and systems:

Authentication in LIMS:

Third-Party MFA Solutions: Some systems integrate MFA through SSO providers like Google or Okta. While convenient, this creates reliance on external services, which may fail or be compromised. Biometric Authentication: A few advanced systems use biometrics (e.g., fingerprint or facial recognition), but these are expensive to implement and require specialized hardware.

Encryption Techniques:

Most existing LIMS systems use encryption for data transmission (e.g., HTTPS), but fail to implement field-level encryption for data at rest. Some systems use database-native encryption, which can be difficult to customize or integrate with application logic.

Role-Based Access Control (RBAC):

Several systems employ RBAC to restrict user access based on roles (e.g., admin, technician), but these implementations often lack flexibility and fine-grained permissions.

Audit Logging:

While audit logging is standard in modern systems, many LIMS implementations fail to provide detailed logs, making it difficult to track unauthorized or suspicious activities.

Open-Source LIMS:

Open-source projects, such as LabKey and Bika LIMS, offer basic functionality but require significant customization to meet security requirements. Many of these systems are built on outdated frameworks, limiting their compatibility with modern tools.

Gaps in Existing Solutions

From our review of related work, we identified the following gaps:

Lack of Comprehensive Security:

Existing systems often address security piecemeal, such as focusing on MFA but neglecting encryption, or vice versa.

Dependency on External Tools:

Reliance on third-party MFA or encryption providers increases the risk of system downtime or external breaches.

Limited Scalability:

Many solutions are tailored for specific use cases and lack the flexibility to adapt to different lab environments.

Complexity for End Users:

Security implementations, where present, are often difficult for users to navigate, leading to poor adoption.

Defining the Solution

Our project aims to address these gaps by designing a secure LIMS-like system with the following features:

- Integrated Security: MFA and field-level encryption are built into the core design, ensuring seamless functionality.
- Custom MFA Implementation: A native MFA workflow eliminates dependency on external providers.
- Field-Level Encryption: Sensitive fields are encrypted at rest using AES-256 encryption, ensuring data confidentiality even in the event of a breach.
- Accountability: Robust audit logs track all user actions for transparency and compliance.
- User-Friendly Interface: A clean and responsive design ensures security measures are intuitive for all users.

3. METHODOLOGY

Our project, a secure Laboratory Information Management System (LIMS), was developed using a modular and secure approach with the Django web framework. Below, we describe the design, implementation, and components of our methodology in detail, highlighting the technical aspects and steps taken to achieve the system objectives.

A. System Architecture

The system is structured around the Django web framework, emphasizing modularity and scalability. The architecture is divided into the following key components:

Core Application (myLIMS):

- Manages the overall settings, configurations, and middleware for the project.
- Files include settings.py, urls.py, asgi.py, and wsgi.py.

Samples Application (samples):

- Handles the core CRUD operations for managing sample data.
- Ensures encryption and decryption of sensitive fields during these operations.

Multi-Factor Authentication (MFA):

- Provides an additional layer of security for user authentication using time-based OTP (TOTP).
- Includes custom middleware to enforce MFA.
- Each module is organized into folders and files to maintain clarity, reusability, and ease of maintenance.

B. Technologies and Tools Used

- Framework: Django 5.1.3
- Authentication:
- Built-in Django authentication for username and password.
- Extended with django-otp and django-otp.plugins.otp-totp for time-based OTP.
- Encryption:
- Sensitive fields are encrypted using django-encrypted-model-fields, which employs AES-256 encryption.
- The FIELD-ENCRYPTION-KEY is securely stored and used for seamless encryption and decryption during CRUD operations.
- Database: SQLite for local development and testing.
- Frontend: Bootstrap for responsive and user-friendly CRUD forms.
- Middleware: Custom middleware (EnforceMFAMiddleware) to enforce MFA on authenticated users.

C. Key Functionalities and Features

User Authentication

The project employs Django's built-in authentication system, extended with MFA for added security:

Login Workflow:

- Users log in using their credentials (username and password).
- Post-login, users are redirected to an MFA verification page where they provide a time-based OTP.

Custom Middleware:

- EnforceMFAMiddleware ensures that MFA is enforced for all authenticated users.

Password Management:

- Includes password reset and recovery functionalities using Django's authentication views.

Data Encryption

- Sensitive fields, such as sample names and types, are encrypted before being stored in the database.
- AES-256 encryption is implemented through django-encrypted-model-fields, ensuring data is protected even if the database is compromised.
- Decryption is performed automatically during data retrieval, allowing seamless integration into CRUD operations.

CRUD Operations

- Full implementation of Create, Read, Update, and Delete functionalities for managing sample data.
- These operations are secured by integrating encryption at the database level.
- User-friendly forms for CRUD operations were built using Bootstrap to enhance the interface.

Audit Logging

- Every user action, such as creating, updating, or deleting a sample, is logged for accountability.
- Logs are accessible to administrators through a secured interface.

D. Folder and File Structure

Core Folder: myLIMS

Purpose: Manages project-wide configurations.

Key Files:

- settings.py: Includes database configurations, installed apps, middleware, and encryption key.
- urls.py: Routes URLs to respective applications (e.g., samples, mfa).
- wsgi.py and asgi.py: Handle web server and asynchronous gateway configurations.

MFA Folder: mfa

Purpose: Implements Multi-Factor Authentication.

Subfolders:

- -pycache- and migrations: System-generated files for Python and database schema changes.
- templates/mfa: Contains HTML templates for MFA views.

Key Files:

- middleware.py: Custom middleware for enforcing MFA.
- views.py: Implements OTP generation and verification.

Samples Folder: samples

Purpose: Manages sample-related CRUD operations and encryption.

Key Files:

- models.py: Defines sample data schema with encrypted fields.
- views.py: Implements logic for CRUD operations and integrates encryption.
- templates: Contains HTML templates for CRUD forms.
- Database: The db.sqlite3 file stores all application data locally.

Generated Files

-pycache-: Stores precompiled Python files for faster execution.

E. Workflow and Development Phases

Setup and Configuration:

Initialized the Django project using django-admin startproject. Configured settings.py with required apps (samples, mfa, django-otp) and middleware.

Authentication Implementation:

Integrated Django's authentication system. Extended with TOTP-based MFA using django-otp and custom middleware.

Encryption Integration:

Defined sensitive fields in the models.py file with AES-256 encryption using django-encrypted-model-fields. Ensured seamless encryption and decryption during CRUD operations.

UI Design:

Built responsive forms using Bootstrap for CRUD operations. Integrated MFA verification templates for a cohesive user experience.

Testing and Debugging:

Tested the system locally using SQLite. Debugged compatibility issues and ensured smooth encryption workflows.

4. IMPLEMENTATION

The implementation of the secure Laboratory Information Management System (LIMS) was carried out in a structured and modular manner using the Django web framework. This section details the technical aspects of the implementation, focusing on authentication, encryption, CRUD operations, and system architecture.

A. System Configuration and Initialization

The project was initiated using the Django 5.1.3 framework, chosen for its scalability, modular design, and robust security features. The myLIMS project folder serves as the core configuration hub, containing critical files such as settings.py, which manages middleware, installed apps, and encryption keys. The system was configured to use SQLite as the database during development for ease of setup and rapid prototyping.

Middleware was a key focus area during initialization. The middleware stack includes security features such as django middleware security. SecurityMiddleware for HTTPS redirection and django-otp.middleware.OTPMiddleware for managing OTP workflows. A custom middleware, EnforceMFAMiddleware, was developed to ensure users authenticate through Multi-Factor Authentication (MFA) after logging in.

B. User Authentication and Multi-Factor Authentication

The Django authentication framework was extended to incorporate MFA, enhancing security beyond single-factor authentication. After a user logs in with their username and password, they are redirected to an MFA verification page. The MFA implementation uses Time-based One-Time Passwords (TOTP), facilitated by the django-otp and django-otp.plugins.otp-totp libraries.

The mfa application manages this workflow:

- TOTP Generation: The backend generates a time-sensitive OTP using the shared secret stored for each user. The user must provide this OTP to complete authentication.
- Middleware Enforcement: The EnforceMFAMiddleware intercepts user requests post-login and redirects users without verified MFA to the verification page, ensuring that all actions are protected.

Custom templates in the templates/mfa directory provide a user-friendly interface for entering and verifying OTPs, ensuring seamless interaction with the system.

C. Data Encryption and Secure Storage

Sensitive data fields, such as sample names and types, are encrypted before storage using AES-256 encryption. The encryption is implemented via the django-encrypted-model-fields library, which integrates directly into Django's ORM. This approach ensures that encryption and decryption processes are seamless during CRUD operations.

The FIELD-ENCRYPTION-KEY, a base64-encoded key, is securely defined in the settings.py file. This key is critical for encryption and decryption and is never exposed to unauthorized access. During runtime:

- Data is encrypted before being written to the database, ensuring that sensitive fields remain unreadable in plaintext.
- When data is retrieved, decryption occurs automatically, allowing applications to function without requiring explicit decryption logic.

This implementation protects data at rest, ensuring confidentiality even if the database is compromised.

D. CRUD Operations for Sample Management

The samples application implements the Create, Read, Update, and Delete (CRUD) functionalities for managing laboratory samples. The models.py file defines the database schema, with encrypted fields for sensitive data. For example, the Sample model includes fields such as name and sample-type, marked as encrypted.

The views.py file orchestrates CRUD operations:

1. Create: Users can input sample details using a Bootstrap-enhanced form. Submitted data is encrypted before storage.
2. Read: Data is retrieved from the database and decrypted on-the-fly, ensuring that sensitive information is presented in plaintext only to authorized users.
3. Update: The system allows users to modify existing records. Updated fields are re-encrypted before storage to maintain security.
4. Delete: Users can delete samples, with the action logged in the audit trail for accountability.

The templates/samples directory provides responsive HTML forms for these operations, ensuring a user-friendly interface.

E. Audit Logging and Accountability

The system includes a comprehensive audit logging mechanism to track user actions. Logs are stored in the database, detailing events such as login attempts, sample creation, and updates. These logs provide transparency and support compliance with regulatory requirements.

Administrators can view these logs via a secure interface, with data displayed in a structured format using Django's admin panel. Logs include details such as:

- User ID performing the action.
- Type of action (e.g., "Created sample", "Updated sample").
- Timestamp of the action.

F. UI Enhancements

The user interface was developed using Bootstrap to ensure responsiveness and usability. CRUD forms, login pages, and MFA verification pages are styled to provide a consistent experience across devices. Templates were designed to minimize clutter and guide users through complex workflows, such as MFA verification.

G. System Workflow

The system operates as follows:

User Login: A user enters their credentials on the login page. If the credentials are valid, the system redirects to the MFA verification page.

MFA Verification: The user enters a TOTP generated by an authenticator app. Upon successful verification, access is granted.

Sample Management: Authorized users can create, view, update, or delete sample records. Encryption ensures that sensitive data remains protected throughout these operations.

Audit Logs: Every action is logged, providing a detailed history of user activities.

H. Testing and Debugging

The system was rigorously tested for:

- Authentication Security: Ensuring that unauthorized access is blocked at all stages.
- Encryption Functionality: Validating that sensitive fields are stored as ciphertext and correctly decrypted for authorized use.
- CRUD Accuracy: Confirming that all operations behave as expected while preserving data integrity.
- User Experience: Testing forms and workflows for clarity and ease of use.

Debugging tools included Django's built-in debug mode and structured logging to identify and resolve issues efficiently.

5. OUTCOME AND DELIVERABLES

The secure Laboratory Information Management System (LIMS) developed during this project successfully addressed the core issues of data security and user authentication while maintaining a user-friendly design. This section outlines the final outcomes and deliverables, emphasizing the technical achievements and how they align with the objectives of the project.

A. System security and authentication

One of the most significant outcomes of the project is the implementation of robust security mechanisms. The system incorporates both native authentication and Multi-Factor Authentication (MFA), providing an additional layer of protection. Users must verify their identity through a Time-based One-Time Password (TOTP) after logging in with their credentials. This ensures that even if passwords are compromised, unauthorized access is effectively prevented.

The system also enforces MFA as a mandatory step for all users through custom middleware, demonstrating a commitment to proactive security measures. This enhancement places the developed LIMS ahead of many existing systems that either lack MFA entirely or rely on third-party providers.

B. Data encryption and confidentiality

Another critical outcome is the successful integration of field-level encryption using AES-256 encryption. Sensitive data fields, such as sample names and types, are encrypted before storage in the database. This ensures that even in the event of a database compromise, the encrypted data remains inaccessible to unauthorized individuals.

The encryption mechanism was seamlessly integrated into the system's Create, Read, Update, and Delete (CRUD) operations. Encryption occurs automatically during data insertion, while decryption is performed transparently during data retrieval. This design minimizes user involvement in the encryption process, ensuring a smooth and secure workflow.

C. Functional CRUD operations

The system delivers fully functional CRUD operations for managing laboratory sample data. Users can securely create new sample records, retrieve existing records, update sample details, and delete records when necessary. These operations are supported by a responsive and intuitive interface, which was designed using Bootstrap for enhanced usability.

The integration of encryption into CRUD operations ensures that security is preserved throughout the data lifecycle. This combination of functionality and security allows laboratories to manage sensitive data efficiently without compromising its confidentiality or integrity.

D. Comprehensive Audit Logging

Accountability is a key feature of the system, achieved through detailed audit logging. Every significant user action, such as creating, updating, or deleting a sample, is logged with information about the user, the action performed, and the timestamp. These logs are securely stored and accessible to administrators, providing full traceability of system activities.

Audit logging not only supports internal accountability but also assists laboratories in meeting compliance requirements for data management and security.

E. Improved User Experience

The system was designed with user experience as a priority. The use of Bootstrap in the front-end ensures that the interface is clean, modern, and responsive across various devices. Forms for authentication, MFA verification, and sample management are intuitive, reducing the learning curve for new users.

This focus on usability ensures that the system is not only secure but also practical for day-to-day laboratory operations. Users can navigate the system efficiently without being overwhelmed by its security features.

F. Scalability and Extensibility

The modular architecture of the system enables scalability, allowing it to support larger datasets and more users as laboratory needs grow. Each component, such as the Multi-Factor Authentication (MFA) and encryption mechanisms, is designed to operate independently, making it easier to extend or customize the system in the future.

The system's use of Django ensures that it can be deployed on various platforms and integrated with additional tools or technologies as needed. This makes the developed LIMS a versatile solution that can adapt to evolving requirements.

G. Final Deliverables

https://github.com/surya30461/LIMS.git

The project's final deliverables include:

1. A fully functional, secure LIMS system that integrates authentication, encryption, and sample management.
2. A structured database with sensitive fields protected by AES-256 encryption.
3. A responsive web interface for managing samples, ensuring both usability and security.
4. Detailed audit logs for all user actions, accessible to administrators for monitoring and compliance.
5. A modular and extensible architecture that supports future enhancements and scalability.

6. BIBLIOGRAPHY

The bibliography provides a comprehensive list of references and resources consulted during the development of the secure Laboratory Information Management System (LIMS). These references include technical documentation, libraries, frameworks, academic papers, and online resources that contributed to the design and implementation of the project.

A. Frameworks and Libraries

Django Documentation

- Django Software Foundation. Django Documentation - The Web Framework for Perfectionists with Deadlines. Retrieved from https://docs.djangoproject.com/.
- This resource was essential for understanding Django's architecture, authentication system, and integration of third-party libraries.

django-otp

- Django OTP. Django OTP: Multi-Factor Authentication for Django. Retrieved from https://django-otp-official.readthedocs.io/.
- Provided guidance on implementing Multi-Factor Authentication (MFA) using time-based one-time passwords (TOTP).

django-encrypted-model-fields

- Django Encrypted Fields. django-encrypted-model-fields Documentation. Retrieved from https://github.com/Incuna/django-encrypted-model-fields.
- This library's documentation was critical for integrating AES-256 encryption for sensitive fields.

Bootstrap

- Bootstrap. Bootstrap Documentation: The Most Popular HTML, CSS, and JS Library in the World. Retrieved from getbootstrap.
- Used for designing responsive and user-friendly web interfaces.

SQLite

- SQLite. SQLite documentation. Retrieved from https://www.sqlite.org/docs.html.
- Provided details on database setup, schema design, and query optimization for lightweight development.

B. Security Resources

National Institute of Standards and Technology (NIST)

- NIST. AES-256 Encryption Standard. Retrieved from https://csrc.nist.gov/.
- Referenced for the encryption algorithm used to secure sensitive data in the database.

OWASP

- OWASP Foundation. Top 10 Security Risks and Best Practices. Retrieved from https://owasp.org/.
- Provided insights into secure authentication mechanisms and encryption techniques to mitigate common security vulnerabilities.

C. Academic and Research Papers

1. Smith, A. and Brown, J. (2021). Enhancing Data Security in Laboratory Information Management Systems Using Encryption and Authentication Protocols. Journal of Data Security, 15(4), 123-135.

   Explored methodologies for integrating encryption and authentication into LIMS systems.

2. Kumar, P. and Rajan, S. (2020). Role-Based Access Control and Data Encryption in Modern LIMS Systems. International Journal of Information Security, 18(3), 210-222.

   Discussed role-based access and encryption strategies, providing foundational knowledge for this project.

3. Patel, M., and Desai, R. (2019). Challenges in Securing Laboratory Data: A Case Study on LIMS. Proceedings of the ACM Conference on Data Management, 12(2), 45-56.

   Highlighted security gaps in existing LIMS and emphasized the need for integrated security solutions.

D. Online Tutorials and Blogs

1. Real Python. Implementing Secure User Authentication in Django. Retrieved from https://realpython.com/.

   Helped in understanding Django's authentication flow and customizing it for MFA integration.

2. Dev.to. Django Encryption and Data Security: A Beginner's Guide. Retrieved from https://dev.to/. Offered practical examples of the integration of encryption in Django models.

3. Medium. Using Django OTP for multifactor authentication. Retrieved from https://medium.com/.

   Guided the implementation of TOTP-based MFA with step-by-step instructions.

E. Toold and Development Resources

1. GitHub. django-otp repository. Retrieved from https://github.com/django-otp/django-otp.

   Source code and examples for integrating Django OTP in web applications.

2. GitHub. django-encrypted-model-fields repository. Retrieved from https://github.com/Incuna/django-encrypted-model-fields.

   Provided access to source code and implementation details for field-level encryption.

3. Stack Overflow. Django Security Best Practices. Retrieved from https://stackoverflow.com/.

   Used to troubleshoot implementation issues and gather information on Django security practices.

F. Standards and Compliance

1. General Data Protection Regulation (GDPR)

   European Union. Regulation (EU) 2016/679: General Data Protection Regulation. Retrieved from https://gdpr-info.eu/.

   Referenced for understanding data protection requirements, particularly for handling sensitive laboratory data.

2. Health Insurance Portability and Accountability Act (HIPAA)

   US Department of Health and Human Services. HIPAA Compliance Standards for Data Security. Retrieved from

   Consulted for ensuring that the system is in accordance with healthcare data security standards.