


# Digital Locksmith



Vinitha Puligorla  
Nikhitha Balagani

# Contents

- ❑ Why Digital Locksmith
  - ❑ Primary Features and target Audience
  - ❑ Why Django?
  - ❑ Main features emphasised
  - ❑ Authentication
  - ❑ Confidentiality
  - ❑ Data Integrity
  - ❑ Interactive UI
  - ❑ Backend implementation
  - ❑ Outline of security features Implemented
  - ❑ Technologies, frameworks, and tools used
  - ❑ Conclusion
- 

## Digital Locksmith


True to its name(Digital Locksmith), it functioned like a skilled locksmith but for digital assets. It not only stored various passwords securely but also generated strong, unique passwords for each account.

Now access all their accounts can be managed with ease, needing to remember just one master password. It was as if a magic wand had been waved, relieving them of their password woes.



# Why Digital Locksmith

In the digital era, managing the passwords is critically important for several reasons:


- **Security:** As digital activities increase, so does the risk of cyber threats. Strong, unique passwords for each account are essential to protect against unauthorized access and data breaches.
  - **Increasing Number of Accounts:** Most individuals have multiple online accounts (social media, banking, email, etc.). Remembering complex passwords for each account is challenging, making password management tools a necessity.
  - **Prevention of Identity Theft:** Effective password management helps prevent identity theft. Using unique, complex passwords reduces the risk of personal information being compromised.
  - **Convenience:** Password managers offer a convenient way to store, retrieve, and manage passwords securely, eliminating the need to remember numerous passwords.
  - **Compliance:** Many businesses are required to adhere to strict data security regulations. Efficient password management ensures compliance with these regulations, protecting sensitive customer and business data.
- 

# Primary Features and target Audience

## Primary Features:

- User Authentication: Secure login system for users to access their password vault.
- Password Encryption: Uses robust encryption methods to protect stored passwords, ensuring they are not accessible in plain text.
- Intuitive Interface: A user-friendly interface that allows easy addition, retrieval, and management of passwords.
- Data Integrity and Security: Implements measures to maintain data integrity and protect against unauthorized access or breaches.
- Search and Organization Features: Offers functionalities to search for specific entries and organize passwords efficiently.

## Target Audience:

- General Internet Users: Individuals who manage multiple online accounts and need a convenient way to securely store and manage their passwords.
  - Security-Conscious Individuals: Users who are aware of digital security risks and prefer a secure method for handling their passwords.
  - Small Businesses or Teams: May benefit from this tool to manage shared passwords for various online services and platforms.
- 

# Why Django?

## Security Features:

- Protection Against Common Attacks: Automatically secures against SQL injection, XSS, CSRF, and clickjacking.
- Secure Password Management: Uses hashing for password storage and includes password strength validators.
- HTTPS/SSL Support: Facilitates encrypted data transmission for security.
- Security Middleware: Offers additional security controls and policies.
- Regular Updates: Constantly updated to fix vulnerabilities.

## Scalability:

- Modular Architecture: The MVT pattern allows independent component scaling.
- Efficient Database Handling: ORM supports efficient querying and scalability with large databases.



# Main features emphasised

- Authenticity
- Data Integrity
- Confidentiality

**Authenticity:** Verifying the identity of a user or the genuineness of data.

**Data Integrity:** Ensuring data remains accurate and unaltered throughout its lifecycle.

**Confidentiality:** Protecting information from unauthorized access and disclosure.



# Authentication

Users create accounts by providing credentials (like username and email) and setting up passwords.

During login, the system verifies these credentials against stored data to authenticate users.

Session Management:

Once authenticated, users are granted a session.

Django tracks user sessions to maintain their authenticated state as they interact with the application.

Security Measures for Authentication:

Implementations like CSRF tokens in forms, which Django handles automatically, to prevent unauthorized form submissions.





# Confidentiality

## **Password Protection**

Passwords stored in the database are not kept in plain text. Instead, they are encrypted using strong hashing algorithms (like PBKDF2, bcrypt, or Argon2), which are standard in Django.

This means even if the database is compromised, the actual passwords remain protected.

## **HTTPS/SSL Implementation:**

The project is likely configured to use HTTPS, ensuring that data transmitted between the user's browser and the server is encrypted.

This secure data transmission prevents eavesdropping and man-in-the-middle attacks.

## **User-Specific Data Access:**

The application ensures that users can only access their own password data.

Proper session management and authentication checks prevent one user from accessing another user's data.

## **Secure File Handling and Storage:**

If the application handles files or other sensitive data, these would also be securely stored and managed.

Measures would be taken to ensure that files are not accessible to unauthorized users.

## **Environment Variable for Sensitive Data:**

Critical information, like database credentials and secret keys, are stored in environment variables, not in the codebase, to prevent exposure.



# Data Integrity

## **Django's ORM (Object-Relational Mapping):**

Django's ORM acts as a layer between the application and the database, ensuring that data is correctly stored and retrieved. It prevents SQL injection attacks, which can compromise data integrity, by using parameterized queries.

## **Form Validation:**

Django forms come with built-in validation that checks for erroneous or malicious data before it's saved to the database. This includes type checking, length validation, and custom validation rules as needed.

## **Model Constraints:**

Django models can be configured with various constraints (such as unique, max\_length, etc.) to ensure data consistency and adherence to business rules.

## **Transactions:**

Django supports database transactions, ensuring that a series of database operations either complete entirely or not at all. This is crucial for maintaining data consistency, especially in the context of operations that involve multiple steps or updates.

## **Data Backup and Recovery Strategies:**

While not always part of the Django framework itself, a robust data backup and recovery plan is essential for maintaining data integrity. Regular backups and a clear recovery plan ensure that data can be restored to a consistent state in the event of corruption or loss.

## **Input Sanitization:**

Ensuring that all user input is sanitized before being processed or stored helps prevent issues like script injections, which can corrupt or alter data.



# Interactive UI

The UI of Digital Locksmith is designed in such a way that it is very easy for the user to understand how each of the pages in the application are used.

Login Screen: Mainly focuses on register and login functionalities

Home Page: To add new passwords and view them.

Add Password: Navigation panel option to add a new password into the system.

Manage Password: To view existing passwords that are added with details

Search: To search for a specific password related to an app

Logout: To securely logout of the application



# Backend implementation

Django's MVT Framework is used.

Database Models: Django uses ORM to map model classes and the database tables.

Database setup: Django supports various databases like MySQL, PostgreSQL and oracle. We have used PostgreSQL for the database. The settings.py contains database configuration details.

Database Migrations: Django manages database schema through migrations using makemigrations and migrate commands.



# Outline of security features Implemented

**Password Hashing:** Uses Django's built-in password hashing and validation mechanisms to securely store user passwords.

**HTTPS/SSL Implementation:** Ensures secure communication between client and server to protect data in transit.

**User Authentication and Session Management:** Leverages Django's authentication system to manage user sessions and credentials securely.

**CSRF Protection:** Employs Django's CSRF tokens in forms to prevent cross-site request forgery attacks.

**SQL Injection Prevention:** Utilizes Django's ORM, which inherently protects against SQL injection attacks.

**Regular Security Updates:** Keeps Django and its dependencies up-to-date to patch vulnerabilities.

**Input Validation:** Enforces strict input validation to prevent XSS attacks and ensure only valid data is processed.

**Secure Configuration:** Uses environment variables for sensitive configurations (like secret keys) to avoid hardcoding them in the source code.

**Data Encryption:** Encrypts sensitive data, particularly passwords, both at rest and in transit.



# Technologies, frameworks, and tools used

- Django Framework: The primary web framework used for the backend. It handles routing, ORM, authentication, and more.
- Python: The programming language in which Django is written and the primary language used for the project.
- Database Technologies: PostgreSQL: Likely choices for production databases due to their robustness and scalability.
- Front-End Technologies:
  - HTML/CSS: For structuring and styling the web pages.
  - JavaScript: For adding interactivity to the web pages.
  - Bootstrap or similar CSS frameworks: For responsive design.
- Virtual Environments: Like venv or virtualenv for Python dependency management.
- Security Tools:
  - SSL Certificates: For HTTPS implementation.
  - Security plugins or middleware specific to Django.



# Conclusion


**Leveraging Django's Strengths:** The project effectively utilizes Django's powerful features, including its ORM, security mechanisms, and user-friendly templating system, to create a reliable password management tool.

**Focus on Security:** With features like hashed password storage, CSRF protection, and potential for SSL implementation, the project prioritizes the security of user data, a critical aspect given its nature.

**Ease of Use:** The intuitive UI design ensures that the application is accessible to users with varying levels of technical expertise, promoting wider adoption.

**Scalability and Maintainability:** The use of Django, along with other modern technologies, ensures that the project is not only scalable but also maintainable and adaptable to future enhancements.

**Future Growth:** The project has room for further development, such as implementing two-factor authentication, cloud synchronization, and cross-platform support, which would greatly enhance its functionality.



ANY QUESTIONS?

