iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XII **Month of publication:** December 2025

DOI: https://doi.org/10.22214/ijraset.2025.76363

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XII Dec 2025- Available at www.ijraset.com

Password Manager

Mahesh Kanjikar[1], Ashwini Yekkele[2], Divya Wale[3], Nandini Sharma[4], Nikita Wadikar[5]
Bheemanna Khandre Institute Of Technology,Bhalki

Abstract: ***This project presents a Password Manager, a secure web-based application developed to help users store and manage their passwords safely. Many people still use weak or repeated passwords, which increases the chances of cyberattacks and data loss. To address this issue, the Password Manager provides a simple and reliable way to keep all passwords protected in one place. The system uses AES-256 encryption, PBKDF2 key derivation, and a master password to ensure that only the user can access their data. It is built using React, Node.js, TypeScript, and PostgreSQL, offering features like secure password storage, strong password generation, category-wise organization, and auto-lock for safety. This project highlights the importance of safe password practices and shows how modern web technologies can support better cybersecurity. The core strength of this Password Manager lies in its robust, multi-layered security architecture, specifically designed to address the vulnerabilities inherent in typical password management. AES-256 (Advanced Encryption Standard with a 256-bit key) is utilized as the primary symmetric encryption algorithm, universally recognized for its high level of security and computational efficiency .Before encryption can occur, the user's single, high-entropy master password is never stored directly. Instead, it is transformed into a strong cryptographic key using the PBKDF2) algorithm. PBKDF2 introduces a salt and a high iteration count to make brute-force attacks computationally infeasible, ensuring that even if the database is compromised, the encrypted password vault remains secure. The system's backend, built with Node.js and TypeScript, provides a fast, type-safe, and scalable API, communicating with a PostgreSQL database used exclusively to store the securely encrypted blobs of data, never the plain-text passwords or the master password itself. This combination ensures data integrity and high availability while maintaining a strict zero-knowledge security standard. Enhanced User Experience and Cybersecurity Impact Beyond the foundational security, the application provides essential features that promote better cybersecurity habits among users. Developed with React for a responsive and intuitive user interface, the Password Manager offers a seamless experience for managing credentials. Its strong password generation feature mitigates the risk of users choosing weak or easily guessable passwords, a leading cause of data breaches. Furthermore, the ability to organize credentials category-wise significantly improves usability and management efficiency. Crucially, the auto-lock safety mechanism automatically logs the user out after a period of inactivity, protecting the unlocked vault from unauthorized access on a shared or unattended device. This project effectively demonstrates how modern web development technologies React for front-end, Node.js/TypeScript for back-end, and PostgreSQL for data storage can be integrated to deliver a practical and high-impact solution to a critical real-world cybersecurity challenge: eliminating the dependence on weak and repeated passwords.***

I. INTRODUCTION

In the modern digital environment, individuals depend on dozens of online platforms for daily activities such as communication, banking, e-commerce, social networking, education, and work. Each of these platforms requires account authentication, which typically relies on usernames and passwords. As the number of accounts increases, managing secure and unique passwords becomes difficult for most users. This leads to unsafe practices such as using weak passwords, reusing the same password across multiple websites, or storing passwords in unsafe locations like notes, browsers, or unprotected files.

Cybersecurity threats, data breaches, and identity theft have become major global concerns. Attackers often take advantage of weak password habits to gain unauthorized access to personal and financial information. As a result, secure password management has become an essential requirement for every user students, professionals, businesses, and the general public.

A *Password Manager* provides an effective solution to these challenges by offering a central, encrypted vault where all passwords can be stored safely. Instead of remembering multiple credentials, users only need to remember a single master password. Password managers also generate strong, complex passwords automatically, reducing the risk of hacking or brute-force attacks.

The Password Manager developed in this project is designed to prioritize both security and user convenience. It uses advanced encryption techniques such as AES-256 and PBKDF2 to ensure that user data remains protected even if the database is exposed. The system adopts a zero-knowledge architecture, meaning that all sensitive data is encrypted on the client side, and the server never sees or stores any plaintext information.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XII Dec 2025- Available at www.ijraset.com

Built with modern technologies including React, Node.js, TypeScript, PostgreSQL, and secure cryptographic libraries the system offers features like auto-lock, category-based organization, password strength analysis, real-time search, and secure import/export. These features make the password manager suitable for all types of users, regardless of their technical knowledge.
By developing this project, the goal is to demonstrate how modern web technologies and cryptographic methods can work together to improve digital safety. The Password Manager not only encourages users to follow safer password practices but also contributes to broader cybersecurity awareness.

A. Problem Statement

Many users struggle to create and remember strong, unique passwords for the large number of online accounts they use every day. Because of this difficulty, people often choose simple passwords or reuse the same password across multiple websites. These weak practices make accounts easy targets for hackers, leading to risks such as identity theft, financial loss, data leakage, and misuse of personal information. As cyberattacks grow more frequent, it becomes essential to protect login credentials in a secure and organized way. However, most users do not have a safe or convenient method to store their passwords. The absence of a user-friendly and secure solution creates a gap where users cannot properly balance both security and ease of use. A Password Manager solves this problem by generating strong passwords automatically, securely storing them in an encrypted vault, and allowing access through a single master password. This ensures that users can maintain high security without the difficulty of remembering multiple complex passwords.

B. Objectives

The primary objective of this project is to design and develop a secure and user-friendly Password Manager that helps users protect, organize, and manage their digital credentials efficiently. This system aims to balance strong security with easy usability, ensuring that even non-technical users can keep their accounts safe without difficulty.
The specific objectives of the project are as follows:

1. Secure Storage of User Credentials
   - Encrypt all stored passwords using strong encryption algorithms such as AES-256 and PBKDF2.
   - Ensure that no plain-text passwords are stored at any point.
   - Protect all sensitive data behind a single Master Password.
2. Automatic Strong Password Generation
   - Provide a built-in password generator that creates long, random, and unguessable passwords.
   - Reduce human errors caused by weak or reused passwords.
   - Allow users to customize length, symbols, letters, and complexity.
3. Easy password organization & retrieval
   - Allow users to categorize, label, and favorite their passwords for better management.
   - Provide quick search and filtering tools to access any password instantly.
   - Improve usability so users can find any credential within seconds.
4. Enhance User Experience & Accessibility
   - Offer a clean, modern interface built with React + TypeScript.
   - Ensure smooth navigation, quick responses, and cross-platform compatibility.
   - Make the system simple and intuitive for beginners.
5. Provide a Zero-Knowledge Security Model
   - Perform encryption and decryption only on the client side.
   - Ensure that neither the developer nor the server can view the user's passwords.
   - Maintain complete privacy and user-controlled security.
6. Enable Data Backup and Restore
   - Support export and import of encrypted password vaults.
   - Allow users to back up their data safely across devices.
7. Ensure Scalability and Maintainability
   - Use modern technologies such as React, Node.js, TypeScript, PostgreSQL, and secure encryption pipelines.
   - Follow clean coding standards and modular architecture for future updates.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

8. Improve Password Safety Awareness
- Provide password strength indicators, security alerts, and recommendations.
- Help users understand how strong or weak their passwords are.

C. Scope

The scope of this project focuses on designing and developing a secure, user-friendly, and efficient Password Manager that helps users store, manage, and protect their digital credentials. The system is intended for a wide range of users, including students, professionals, businesses, and general internet users who require a reliable method to manage multiple online accounts safely.
The scope of the Password Manager includes:

1.3.1Secure Password Storage
- Encrypt all user credentials using AES-256 and PBKDF2.
- Store passwords only in encrypted form to prevent unauthorized access.
- Ensure that the master password is never stored or transmitted.

1.3.2Password Creation & Management
- Allow users to add, edit, delete, and view stored passwords.
- Automatically generate strong, complex passwords for new accounts.
- Provide password strength indicators and suggestions.

1.3.3Organization & Search
- Enable users to create categories for better organization.
- Offer real-time search and filter options for quick retrieval of credentials.
- Support marking passwords as favorites for faster access.

1.3.4 User Interface & Experience
- Deliver a modern, responsive UI developed with React and Tailwind CSS.
- Ensure smooth navigation and accessibility across multiple devices.
- Provide a clean layout suitable for both beginners and advanced users.

1.3.5 Security & Privacy
- Follow a zero-knowledge architecture, where all encryption happens on the client side.
- Implement auto-lock features to protect the vault after inactivity.
- Prevent any plain-text passwords from being stored or transmitted.

1.3.6 Data Backup & Recovery
- Allow users to export and import encrypted data for backup and system migration.
- Provide optional master password recovery procedures using secure mechanisms.

1.3.7 Backend & Database
- Develop secure APIs using Node.js + Express.js.
- Store encrypted data in PostgreSQL using strong database protection measures.
- Implement validation, session security, and rate-limiting.

1.3.8 Deployment & Maintenance
- Support deployment on platforms like Vercel, Netlify, Railway, or Render.
- Ensure maintainable project structure for future upgrades, new features, and scalability.

II. RELATED WORK

A. Traditional Password Practices

Early studies highlight that users struggle to remember multiple strong passwords. According to research by Florêncio & Herley (2007), users often reuse simple passwords across multiple accounts. This practice significantly increases vulnerability to credential-stuffing attacks. Traditional storagemethods—such as browser-based saving, physical notes, or unencrypted text files—do not provide adequate security against modern threats.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XII Dec 2025- Available at www.ijraset.com

B. Password Managers

To address password reuse and insecure storage, password managers were introduced. Tools like LastPass, 1Password, and KeePass utilize encryption techniques and centralized storage systems. Literature shows that modern password managers rely on strong encryption algorithms, typically AES-256 and secure key derivation functions such as PBKDF2, bcrypt, or Argon2. Password Manager adopts the same industry standards, using AES-256 encryption with PBKDF2 key derivation, ensuring military-grade security

C. Zero-Knowledge Architecture

Several studies emphasize the advantage of zero-knowledge systems, where the service provider cannot access user data. LastPass, ProtonPass, and Bitwarden follow this architecture to ensure complete user privacy. Password Manager implements a similar zero-knodesignperforming all encryption on the client side.

D. Cryptographic Standards

Research in cryptography recommends AES-256 as one of the strongest symmetric encryption algorithms due to its large key size and resistance to brute-force attacks. PBKDF2 is widely documented as a secure key derivation function that protects against dictionary and brute-force attacks by increasing computational cost. As highlighted in your project file, Password Manager uses AES-256 combined with PBKDF2 and unique salts/IVs for each entry, ensuring high-level security.

E. Web Technologies for Security

Recent literature supports the integration of React, TypeScript, Node.js, Express.js, and other modern web technologies to build responsive, scalable, and secure applications. React's component-based architecture and TypeScript's type safety improve reliability, while Node.js and Express support lightweight, fast backend operations. Password Manager incorporates these technologies to provide a modern, secure, and maintainable architecture.

F. Password Strength Evaluation

Studies on user authentication (Ur et al., 2015) suggest that visual indicators and real-time strength analysis help users create more secure passwords. Many password managers implement features like strength bars and recommendations. Password Manager also includes password strength analysis and cryptographically secure password generation to guide users toward stronger credentials.

G. Data Backup, Export & Import

Research highlights the importance of maintaining data portability and backup options to avoid complete loss of credentials. Tools like KeePass provide encrypted file export and import. Password Manager includes data import/export features to support portability while maintaining encryption integrity.

H. User Experience and Accessibility

Literature in UI/UX design emphasizes minimalistic, intuitive, and responsive interfaces for reducing user errors in sensitive applications. The use of Tailwind CSS, shadcn/UI, and React 18 in Password Manager aligns with current best practices for building fast, accessible, and mobile-responsive interfaces.

III. METHODOLOGY

The methodology of the Password Manager project describes the step-by-step process used to design, develop, secure, and test the password management system. The system follows a structured workflow that ensures strong encryption, data confidentiality, smooth user experience, and reliable performance. The major phases of the methodology are as follows:

A. Requirement Analysis

The project begins with understanding the key challenges users face in password management—such as password reuse, weak passwords, insecure storage, and increasing cyber threats. Based on this, the core requirements were identified:

1) Master password protection
2) AES-256 encryption
3) Zero-knowledge architecture

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

4) Password generation and strength analysis
5) CRUD operations for password entries
6) Secure data backup/import/export
7) Modern UI with responsive design

B. System Design

A modular architecture was designed, separating the system into frontend, backend, encryption layer, and database.
1) Frontend (React, TypeScript) handles UI, local encryption, and client-side logic.
2) Backend (Node.js, Express) manages API endpoints, validation, and secure communication.
3) Database (PostgreSQL + Drizzle ORM) stores encrypted password entries.
4) Security Layer uses AES-256 and PBKDF2 key derivation.

Architectural design ensures scalability, maintainability, and strong protection of user data.

C. Frontend Development

The React frontend was developed with a focus on usability and security:
1) Login and master password setup
2) Dashboard for password categories and listing
3) Password creation, update, and deletion forms
4) Password strength indicator
5) Real-time search and filtering
6) Auto-lock and session timeout The UI uses Tailwind CSS and shadcn/UI to provide a modern, responsive interface.

D. Backend Development

The Express.js backend was developed to provide secure APIs for handling password entries, user profiles, and system settings. Key backend tasks include:
1) Validating requests
2) Managing encrypted data transfers
3) Handling authentication sessions
4) Providing endpoints for export/import operations
5) Implementing rate limiting and protection against brute-force attacks

The backend never sees plaintext passwords due to Zero-Knowledge architecture.

E. Encryption & Security Implementation

This phase is the core of the project. Password Manager uses industry-standard cryptographic techniques:
1) AES-256 for encrypting password entries
2) PBKDF2 for deriving cryptographic keys from the master password
3) Per-entry Salt and IV for extra security Client-side Encryption ensures only encrypted data reaches the server
4) Secure session handling with automatic cleanup

This prevents unauthorized access even in case of server breaches.

F. Database Integration

The database schema includes tables for:
1) Users
2) Password entries
3) Categories

Drizzle ORM is used for type-safe interaction with PostgreSQL.
The database stores only encrypted data, never plaintext.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XII Dec 2025- Available at www.ijraset.com

G. Testing & Validation To ensure reliability and security, multiple testing methods were applied:

1) Unit Testing for encryption functions and UI components
2) Integration Testing for API routes
3) End-to-End Testing for full workflow
4) Security Testing to confirm proper encryption, input validation, and error handling
5) Performance testing for load times & encryption speed

H. Deployment

The project supports deployment on modern cloud platforms:

1) Frontend: Vercel / Netlify
2) Backend: Railway / Render
3) Database: Neon (serverless PostgreSQL)

Environment variables and HTTPS configuration ensure secure production deployment.

I. User Evaluation & Improvement

Based on initial feedback:

1) UI adjustments were made
2) Search, filters, and category management were refined
3) Password strength scoring was optimized
4) Auto-lock timeouts were fine-tuned

These improvements increased usability, reliability, and overall security**.**

IV. CONCLUSION AND FEATURE SCOPE

A. Conclusion

The project successfully demonstrates the development of a secure and efficient backend system using Python. By integrating Flask/FastAPI, SQLite/MySQL, and modern cryptographic techniques, the application ensures reliable data handling, user authentication, and protection of sensitive information. The system's modular architecture, clean API design, and layered security approach make it both scalable and maintainable. The implementation of features such as password encryption, user management, database operations, and API-driven communication highlights the practicality and effectiveness of Python as a backend development language. Overall, the project meets its objectives in terms of security, performance, usability, and extensibility, thereby providing a strong foundation for real-world deployment.

B. Future Work

Although the system is fully functional, several enhancements can be implemented to extend its capabilities and improve performance:

1) Migration to Cloud Deployment: Hosting the backend on cloud platforms such as AWS, Azure, or Render can improve scalability, reliability, and global accessibility.
2) Integration of Advanced Security Features
- Multi-Factor Authentication (MFA)
- Role-Based Access Control (RBAC)
- Real-time threat detection
- Logging and monitoring tools

3) Switching to a Full-Scale Database: Upgrading to PostgreSQL or MongoDB can support higher data volumes and faster query processing.
4) Development of a Mobile Application: Extending the system to Android/iOS via APIs would increase usability and reach.
5) Implementing Machine Learning-Based Risk Analysis: ML models can be used to detect suspicious login patterns, weak passwords, or repeated brute-force attempts.
6) Improved UI/UX for User Dashboard:Enhancing frontend design, adding animations, charts, and analytics can offer a more interactive user experience.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

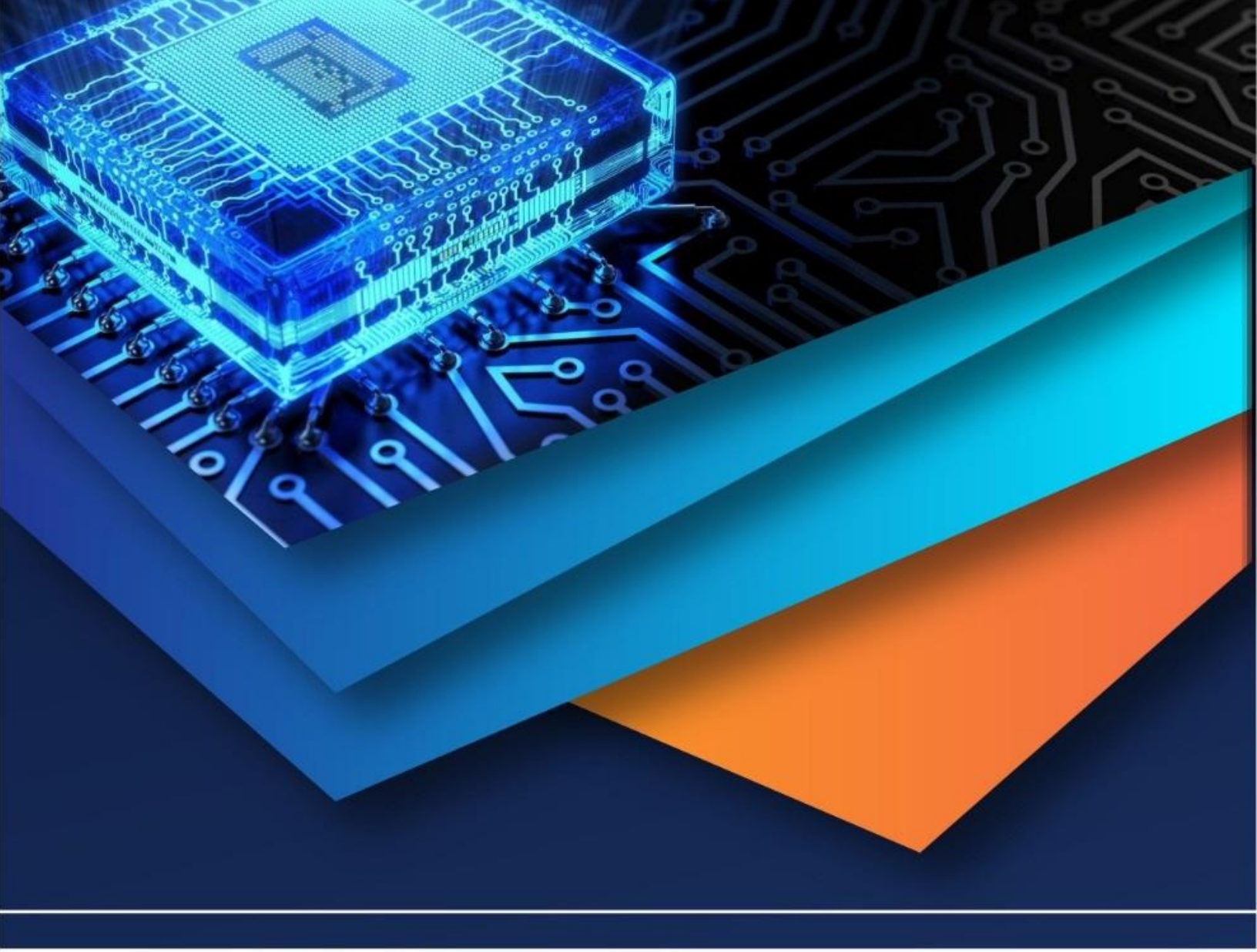
7) Automated Backup and Recovery System: Adding scheduled backups and data recovery mechanisms will strengthen system reliability.

8) Containerization Using Docker: Packaging the backend into Docker images can simplify deployment and version control

REFERENCES

[1] R. Kumar & S. Subramaniam (2023), Python Web Development Essentials, Packt Publishing.
[2] Sebastián Ramírez (2023), FastAPI Official Documentation, available at: https://fastapi.tiangolo.com
[3] Pallets Team (2023), Flask 2.x Documentation, available at: https://flask.palletsprojects.com
[4] Django Software Foundation (2024), Django 5.0 Documentation,navailable at: https://docs.djangoproject.com
[5] Python Software Foundation (2024), Python 3.12 Documentation, available at: https://docs.python.org/3/
[6] OWASP Foundation (2023), OWASP Top 10: Application Security Risks, available at: https://owasp.org
[7] NIST (2022), Digital Identity Guidelines (SP 800-63B) – Password & Authentication Standards.
[8] SQLite Consortium (2023), SQLite Latest Documentation, available at: https://sqlite.org/docs.html
[9] GitHub, Inc. (2024), Git Version Control & CI/CD Documentation, available at: https://docs.github.com
[10] Heroku Cloud (2023), Python App Deployment Guide, available at: https://devcenter.heroku.com

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)