International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Building a Web-Based Password Manager Using the MERN Stack: A Fundamental Approach

Chaithra V S[1], Vinay N[2], Rishikesh Ranjan[3], Raghavendra P[4], Raunak Thakur[5]

[1]Assistant Professor, Sambhram Institute of Technology
[2,3,4,5]UG Students, Sambhram Institute of Technology

Abstract

This paper presents the design and implementation of a basic password manager web application. Built using the MERN stack, the project focuses on providing essential functionalities for managing user credentials. While it lacks advanced security measures like encryption, the project serves as a practical educational tool for understanding full-stack development principles. Inspired by the IEEE paper *Improving Web Application Development Course* by Natheer Khalif Gharaibeh, this research discusses the system architecture, implementation details, limitations, and potential future enhancements.

INTRODUCTION

Password management is an increasingly critical aspect of modern digital life, given the proliferation of online accounts and the risks associated with weak or reused credentials. Cybersecurity professionals consistently highlight the importance of strong, unique pass- words for every account, yet individuals often lack accessible tools to effectively manage them. Password managers address this gap by securely storing and organizing credentials, enabling users to retrieve and use them conveniently.

This research explores the design and development of a basic password manager web application. The project leverages the MERN (MongoDB, Express.js, React, Node.js) stack, offering a streamlined and intuitive platform for storing, retrieving, and managing passwords. While this implementation focuses on fundamental functionalities without en- cryption or advanced security protocols, it serves as a foundational step in understanding full-stack web application development.

The project's core objective is to create a scalable yet straightforward application that demonstrates the seamless integration of frontend and backend technologies. It draws inspiration from the IEEE paper Improving Web Application Development Course by Natheer Khalif Gharaibeh, which emphasizes the educational benefits of hands-on projects in web development curricula. By implementing this project, students and de- velopers can grasp key concepts of system architecture, user interface design, and database management.

This paper also discusses the limitations of the current system, including the absence of encryption and user authentication, and suggests pathways for future enhancements.

In doing so, it aims to bridge the gap between educational projects and real-world appli- cations.

LITERATURE REVIEW

Password managers are widely used tools designed to enhance user security and conve- nience. Existing solutions, such as LastPass and Dashlane, offer features like encryption, password generation, and multi-device synchronization. However, these applications are often complex and may require subscriptions, making them less accessible to beginners and learners.

This project seeks to bridge this gap by creating a simplified version that focuses on the core functionalities of password management. The emphasis is on understanding the MERN stack and the principles of web application development, rather than delivering a production-ready solution.

METHODOLOGY

The methodology adopted for developing the password manager project is structured to ensure a systematic and iterative approach. The development process involved stages of requirement analysis, design, implementation, testing, and deployment.

Requirement Analysis

The initial phase involved identifying key functionalities of the password manager. The requirements were divided into:

Functional Requirements:

- Ability to add, view, update, and delete password entries.
- User-friendly interface for seamless interaction.

Non-Functional Requirements:

- Scalability to handle a growing number of entries.
- Maintainability for future feature enhancements.

Design Phase

The design phase focused on creating an architecture that ensures modularity and scala- bility. Tools like draw.io were used to create system design diagrams, including use-case diagrams and system architecture.

Key design considerations included:

- Modular frontend components for reusability.
- RESTful API design for backend services.
- A document-oriented database schema in MongoDB for flexible data storage.

Implementation Phase

The implementation phase followed the MERN stack:

- **Frontend:** React was used for creating a dynamic and responsive user interface.
- **Backend:** Node.js and Express.js provided the application logic and API end- points.
- **Database:** MongoDB served as the NoSQL database for storing password entries.

Version control was managed using Git, with a branching strategy to handle feature development and bug fixes.

Testing Phase

Testing was carried out to ensure the application met its functional and non-functional requirements:

- **Unit Testing:** Individual components and API endpoints were tested.
- **Integration Testing:** Verified that the frontend, backend, and database commu- nicated

effectively.

- **User Testing:** A small group of users interacted with the application to provide feedback on usability.

Bugs identified during testing were tracked and resolved.

Deployment Phase

The final deployment involved hosting the application on cloud platforms:

- The frontend was deployed on Vercel for fast and reliable content delivery.
- The backend and database were deployed on Heroku and MongoDB Atlas, respec- tively.

Continuous integration and deployment (CI/CD) pipelines were set up to automate testing and deployment.

Future Methodological Enhancements

Given the current limitations, the following enhancements are planned:

- Integration of encryption algorithms for secure password storage.
- Incorporation of multi-factor authentication (MFA) for enhanced user security.
- Transition to containerized deployment using Docker for streamlined scalability.

SYSTEM DESIGN

Architecture

The application uses the MERN stack, comprising:

MongoDB: For storing user credentials in a NoSQL database.

Express.js: For creating RESTful APIs.

React: For building a responsive and interactive frontend.

Node.js: For server-side logic and API handling.

Database Schema

The database includes the following fields:

Service Name: The name of the online service.

Username: The associated username.

Password: The corresponding password.

Data Flow Diagram

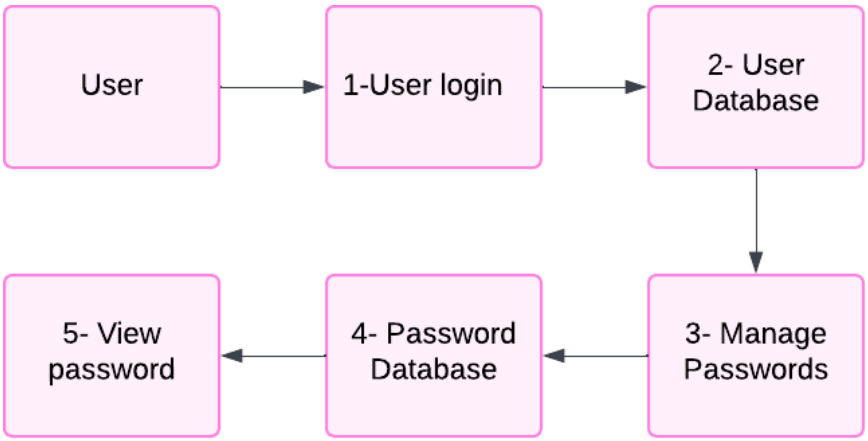

Figure 1: Data Flow Diagram of the Password Manager

IMPLEMENTATION

The implementation of the password manager application was carried out using the MERN stack,

encompassing the development of the frontend, backend, and database. This section outlines the technical details and processes involved in building each com- ponent of the system.

Frontend Development

The frontend was developed using React with Vite for fast builds and a modular compo- nent structure. The main components include:

Login and Registration Pages: Allow users to access the system and register new accounts.

Dashboard: Displays stored passwords and provides options to add, edit, and delete entries.

Form Components: Include input validation to prevent invalid data entry.

Styling was achieved using Tailwind CSS, ensuring a responsive and visually appealing user interface. State management was handled using React Context API to manage user sessions and application state efficiently.

Backend Development

The backend was built using Node.js and Express.js, exposing a RESTful API for com- munication with the frontend. Key API routes include:

- POST /api/auth/register: Handles user registration.
- POST /api/auth/login: Authenticates users and generates JWT tokens.
- GET /api/passwords: Retrieves all stored passwords for a user.
- POST /api/passwords: Adds a new password to the database.
- PUT /api/passwords/:id: Updates an existing password.
- DELETE /api/passwords/:id: Deletes a specific password by ID.

Authentication was implemented using JSON Web Tokens (JWT) to secure API ac- cess. Middleware functions were used for validation and error handling.

Database Schema Design

MongoDB was chosen for its flexibility and scalability. The schema design for the appli- cation includes two primary collections:

Users Collection:

id: Unique identifier for each user.

email: User's email address.

password: Hashed password using bcrypt.

Passwords Collection:

id: Unique identifier for each password entry.

userId: Reference to the user owning the entry.

serviceName: Name of the service (e.g., Gmail, Facebook).

username: Username for the service.

password: Plain text or hashed password (future versions will include encryp- tion).

MongoDB Atlas was used as the database service, ensuring reliable storage and man- agement of data.

Integration and Testing

Integration of the frontend and backend was achieved through the defined API endpoints. The following testing approaches were adopted:

Unit Testing: Conducted for individual React components and backend API routes.

Integration Testing: Ensured seamless communication between the frontend, backend, and database.

Manual Testing: Validated the user experience by simulating common user in- teractions. Testing tools included Postman for API testing and Jest for React component testing.

Deployment

The deployment strategy ensured accessibility and scalability:

Frontend: Hosted on Vercel for fast delivery of static assets.

Backend: Deployed on Heroku with environment variables for secure configura- tions.

Database: Hosted on MongoDB Atlas, with IP whitelisting for security.

Deployment pipelines were set up for automated builds and deployments, reducing the risk of errors during updates.

Security Measures

Although the current version lacks advanced encryption, the following basic security measures were implemented:

Password hashing using bcrypt for user authentication.

JWT-based authentication for API access control.

Input validation to mitigate SQL injection and XSS attacks.Future versions aim to incorporate end-to-end encryption for password storage.

FEATURES AND LIMITATIONS

Features

The password manager application incorporates the following features to provide a user- friendly and functional experience:

User Authentication: Secure login and registration using hashed passwords and JSON Web Tokens (JWT) for session management.

Password Storage: Users can store and manage credentials for multiple services, including service names, usernames, and passwords.

CRUD Operations: Full support for Create, Read, Update, and Delete opera- tions on stored passwords, ensuring complete control over data.

Responsive Interface: A clean, intuitive, and mobile-friendly design using Tail- wind CSS for seamless access on different devices.

Data Segregation: User data is isolated, with access restricted to individual accounts through unique identifiers in the database.

Scalability: Built on the MERN stack, enabling easy scaling and deployment for a growing user base.

API Integration: Modular API endpoints allow future integration with external tools or services.

Deployment Readiness: Hosted on cloud platforms (Vercel, Heroku, and Mon- goDB Atlas) for reliability and accessibility.

Limitations

Despite its current functionality, the application has the following limitations:

Lack of Advanced Encryption: Passwords are stored in plaintext or hashed format but lack end-to-end encryption, making them vulnerable if the database is compromised.

No Multi-Factor Authentication (MFA): User accounts are protected by a single-factor authentication mechanism, reducing overall security.

Limited User Customization: Users cannot customize features such as folder organization, tagging, or automated password generation.
Basic Error Handling: Error messages are generic and may not provide adequate feedback for troubleshooting.
No Offline Access: The application requires an active internet connection to interact with the cloud-based backend and database.
Scalability Constraints: Current deployment may face limitations when handling large-scale traffic without significant architectural improvements.
Security Gaps: While basic measures are implemented, protection against more sophisticated attacks (e.g., brute force, token hijacking) is limited.
Lack of Versioning: Updates to stored passwords do not maintain a history or version control for audit purposes.

Future Enhancements

To address the above limitations and improve functionality, the following enhancements are planned for future iterations:

- Integration of AES encryption for secure password storage.
- Implementation of Multi-Factor Authentication (MFA) to enhance account security.
- Support for folder organization and tagging to improve password management.
- Development of a browser extension for autofilling credentials.
- Introduction of a password generator with adjustable complexity settings.
- Comprehensive logging and audit trails for password updates.
- Advanced error handling to provide detailed and user-friendly feedback.

TESTING AND VALIDATION

The application underwent unit and integration testing to ensure functionality. Test cases included:

- Adding passwords with valid and invalid inputs.
- Retrieving stored passwords.
- Deleting passwords and verifying database updates.

FUTURE WORK

To enhance the application, the following improvements are proposed:

- Implementing AES encryption for secure password storage.
- Adding user authentication and role-based access control.
- Integrating features like password strength analysis and auto-generation.
- Expanding the UI for better user experience.

CONCLUSION

This project demonstrates the development of a basic password manager using the MERN stack. With additional features and security measures, the application could evolve into a robust password management solution.

References

1. Natheer Khalif Gharaibeh, *Improving Web Application Development Course*, IEEE.

2. MERN Stack Documentation: https://www.mongodb.com/mern-stack
3. React Documentation: https://reactjs.org/docs/getting-started.html