**Project Report**

**Overview**

This Node.js project integrates several core features including password encryption, user authentication, session management, dynamic templating with EJS, and database management with MariaDB. The goal was to create a fully functional web application that supports user sign-up, login, and logout, while storing data securely and efficiently.

**Project Features Implemented**

**1. Nodemon for Development**

* **Objective**: Simplify development by automatically restarting the server on code changes.
* **Implementation**: Integrated nodemon to monitor changes in the project files and restart the server automatically, improving the development cycle.
* **Benefit**: Developers no longer need to manually restart the server after making code changes, improving efficiency.

**2. Dynamic Templating with EJS**

* **Objective**: Use a templating engine to generate dynamic HTML pages with embedded JavaScript.
* **Implementation**: Integrated **EJS** for generating HTML content dynamically based on the data passed from the server. Utilized **partials** to manage common components (like headers, footers, etc.) across multiple views, reducing code repetition.
* **Benefit**: Simplified the HTML code by reusing components and making it dynamic, which is easier to maintain.

**3. MariaDB Integration**

* **Objective**: Implement a robust, relational database to store user data and other application-specific information.
* **Implementation**: Configured **MariaDB**, a MySQL-compatible database, to store and manage application data. Utilized the pluggable storage engine architecture (e.g., InnoDB) for better transaction and foreign key support.
* **Benefit**: Enabled efficient data storage with support for transactions and relationships between tables.

**4. User Authentication and Password Management**

* **Objective**: Implement secure authentication mechanisms for user login.
* **Implementation**: Used bcryptjs to hash passwords before storing them in the database. Implemented user authentication by comparing entered passwords with the hashed ones during login.
* **Benefit**: Enhanced security by ensuring that passwords are never stored in plain text, thus preventing unauthorized access.

**5. User Sessions with express-session**

* **Objective**: Manage user sessions for logged-in users to ensure secure access to protected routes.
* **Implementation**: Integrated express-session to handle session management, storing user information on the server and associating it with a unique session ID.
* **Benefit**: Ensured that users remain logged in across multiple requests, providing a seamless experience while preventing unauthorized access to restricted areas.

**6. User Logout and Session Destruction**

* **Objective**: Allow users to securely log out of the application by terminating their session.
* **Implementation**: Created a logout route that calls req.session.destroy() to destroy the session and clear the session cookie.
* **Benefit**: Ensured users' sessions are terminated when they log out, preventing unauthorized access after they leave the application.

**Technologies Used**

* **Node.js**: Server-side JavaScript runtime used for handling HTTP requests and routing.
* **EJS**: Templating engine for rendering dynamic HTML pages.
* **MariaDB**: Relational database management system for data storage.
* **bcryptjs**: Library for hashing and comparing passwords securely.
* **express-session**: Middleware for managing user sessions.

**Challenges Faced**

* **Password Security**: Ensuring that passwords were stored securely was a challenge, but with the use of bcryptjs, we were able to handle password encryption and comparison effectively.
* **Session Management**: Managing sessions and ensuring users are logged in across different routes was initially tricky. However, using express-session, this process became streamlined and reliable.

**Future Enhancements**

* **Password Reset Functionality**: Implement functionality for users to reset their forgotten passwords using email verification.
* **User Authorization**: Introduce role-based access control (RBAC) to restrict access to certain routes based on user roles.
* **Two-Factor Authentication**: Add an extra layer of security by implementing two-factor authentication for user logins.

**Conclusion**

This project successfully integrates several key features required for modern web applications, including secure user authentication, session management, dynamic templating, and relational database storage. The application is now robust and secure, with user login functionality, encrypted password storage, and session management in place.