

CW2 Report for ST6005CEM Security

Submitted By:

Amit Shrestha

Coventry ID: 11782302

College ID: 210185

Submitted To:

Arya Pokharel

Module Leader

Acknowledgement

I would like to express my sincere gratitude to Arya Pokharel, our respected Security Module Leader, for his outstanding guidance and instruction throughout the module. Arya's deep expertise and dedication to creating a supportive learning environment have greatly enhanced our comprehension of security principles. His consistent support, approachability, and commitment to our academic growth have been invaluable, and I am truly thankful for the positive influence he has had on our educational journey. Under Arya's leadership, I have acquired valuable knowledge and skills that will undoubtedly benefit my future endeavors in the field of security. Thank you for your exceptional mentorship and dedication to our academic success.

Table of Contents

Introduction.....	6
System Design Overview.....	7
Features of the system.....	8
Limitations on Passwords	8
Enhanced Authentication via Precise Match Verification Design Method	9
Application of Logic	9
Unique Email	10
Design Approach	10
Implementation	10
Limited Login Attempts.....	12
Design Approach	12
Password Hashing	13
Design Approach	13
Implementation	13
Incorporation of Two-Factor Authentication (2FA)	14
Design Approach	14
Audit Logs	14
Design Approach	14
Implementation	15
User Access Levels	15
Design Approach	15
Implementation	15
Session Management.....	16
Design Approach	16
Feature for encrypting data	17
Design Approach	17
Implementation	17
Conclusion	18
References.....	19

Table of Figures

Figure 1 Limitations on passwords	8
Figure 2 Exact Match Validation	9
Figure 3: Unique Login.....	10
Figure 4: Password Hashing	13
Figure 5: Session Management.....	16

Abstract

This research explores the detailed design and implementation of security measures within a travel diary application that incorporates social media integration, using the MERN stack alongside Next.js. The project prioritizes the protection of cherished memories and personal information, reflecting the growing significance of digital security in today's interconnected world. The system design overview emphasizes the strategic selection of the MERN stack and Next.js, highlighting their combined effectiveness in strengthening the application's security framework. The paper elaborates on various security features such as stringent password requirements, exact match validation, unique email processing, restricted login attempts, password encryption, two-factor authentication, audit trails, user access control, session management, and data encryption. Each feature is thoroughly described, providing insights into both the design strategy and its practical application. The study concludes by emphasizing the integral connection between e-commerce and security, demonstrating how these advanced security protocols help build a robust and trustworthy digital platform for users participating in online activities. The research underscores the importance of user trust and data protection as the cornerstone for establishing a secure and dependable environment, effectively addressing the ongoing challenges posed by the modern digital landscape. By implementing these comprehensive security measures, the application not only safeguards user data but also ensures a seamless and secure experience for users as they engage with the platform.

Introduction

In this day and age of digital technology, when people actively engage in virtual experiences and form social connections, memory preservation becomes absolutely essential. This highlights how crucial it is to have security safeguards in place to protect priceless experiences shared via a trip journal app that integrates with social media. This paper offers a thorough examination of a project focused on security inside a trip journal application. The program's main goal is to capture and preserve vacation experiences while making it simple for users to share them on social media. The technology basis for this project is going to be the MERN stack combined with Next.js. It will create a strong, secure digital environment.

The travel log application was given precedence over other apps, and social media integration was added since it is a flexible platform that lets users share their experiences with others and preserve special memories. Users rely on the platform to safeguard their private and sensitive information as well as the safety of their interactions inside the social media sphere. With the support of Next.js, the MERN stack was carefully selected to offer a contemporary and adaptable technology stack that effectively satisfies functional and security requirements inside the context of a social media-integrated travel application. This paper provides a comprehensive analysis of authentication and encryption methods designed to meet the unique requirements of a social media-integrated travel log application. Moreover, it illuminates the creative efforts aimed at creating and assessing software programs that effectively tackle the present security issues concerning the preservation, distribution, and interaction with travel memories on an online platform that integrates social media features.

System Design Overview

In this day and age of digital technology, when people actively engage in virtual experiences and form social connections, memory preservation becomes absolutely essential. This highlights how crucial it is to have security safeguards in place to protect priceless experiences shared via a trip journal app that integrates with social media. This paper offers a thorough examination of a project focused on security inside a trip journal application. The program's main goal is to capture and preserve vacation experiences while making it simple for users to share them on social media. The technology basis for this project is going to be the MERN stack combined with Next.js. It will create a strong, secure digital environment. By intercepting and authenticating incoming requests, middleware reduces typical vulnerabilities like XSS. By providing protections against SQL injection vulnerabilities, the Mongoose ORM enhances application security by default. Next.js is a framework built on top of React that leverages server-side rendering to reduce data exposure and successfully mitigate security breaches. As a runtime environment, Node.js greatly improves security and efficiency. Because of its event-driven architecture, the system is less vulnerable to Denial of Service (DoS) attacks, and vulnerabilities are promptly fixed thanks to the vibrant development community. Through the deliberate integration of the MERN stack and the implementation of stringent security measures in each component, the project creates a strong and secure base that effectively guards against potential security vulnerabilities.

Features of the system

Limitations on Passwords

Design Method for Complexity of Passwords Because user-selected passwords are somewhat predictable, limitations must be added to improve their effectiveness. To safeguard the integrity of the system against potentially harmful data, server-side data validation for password complexity is necessary, especially following client-side validation.

```
// Password complexity check
const passwordRegex =
  /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/;
if (!passwordRegex.test(password)) {
  return res.status(400).json({
    success: false,
    message:
      "Password must include at least one uppercase letter, one lowercase letter, one number, and one special character.",
  });
}
```

Register

Username
sam

Email
sambahaduramit@gmail.com

Password

Confirm Password

Register

Password must include at least one uppercase letter, one lowercase letter, one number, and one special character.

Figure 1 Limitations on passwords

The server-side "passwordRegex" application uses regular expressions to assess the length and specific criteria (capital letters, lowercase letters, digits, and special characters) of the password in order to guarantee its complexity. The system's overall resilience is increased by maintaining thorough documentation for errors, which results in a detailed JSON response that makes it easier to identify and fix vulnerabilities.

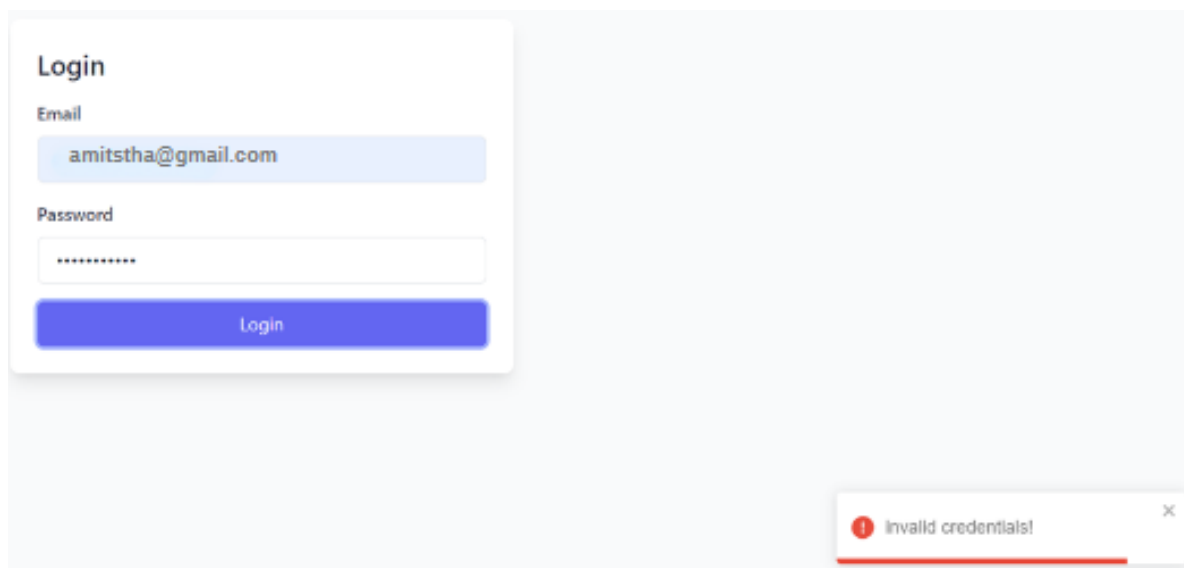
Enhanced Authentication via Precise Match Verification

Design Method

To further improve password security, the system now requires an exact match to be made between the password or email entered and the relevant information stored in the database. Passwords containing personal information are not allowed thanks to the stricter validation procedure. The approach discourages the use of readily predictable passwords and significantly reduces the risk of unauthorized access by using passwords that contain identifying information by employing a case-sensitive comparison.

Application of Logic

The enhanced logic comprises a direct comparison between the user-provided email address or password and the database's recorded data. In the event of a mismatch, an incorrect credential is detected and a response code of 400 is generated. By using a strict validation process, the system is better protected against vulnerabilities that could result from using personal information in passwords.



The image shows a web-based login interface. On the left, there is a white login card with the title 'Login'. It contains two input fields: 'Email' with the value 'amitstha@gmail.com' and 'Password' with masked characters '*****'. Below these fields is a blue 'Login' button. To the right of the card is a large, light gray rectangular area. In the bottom right corner of this area, there is a red error message box that says 'Invalid credentials!' with a close button (X) in the top right corner.

Figure 2 Exact Match Validation

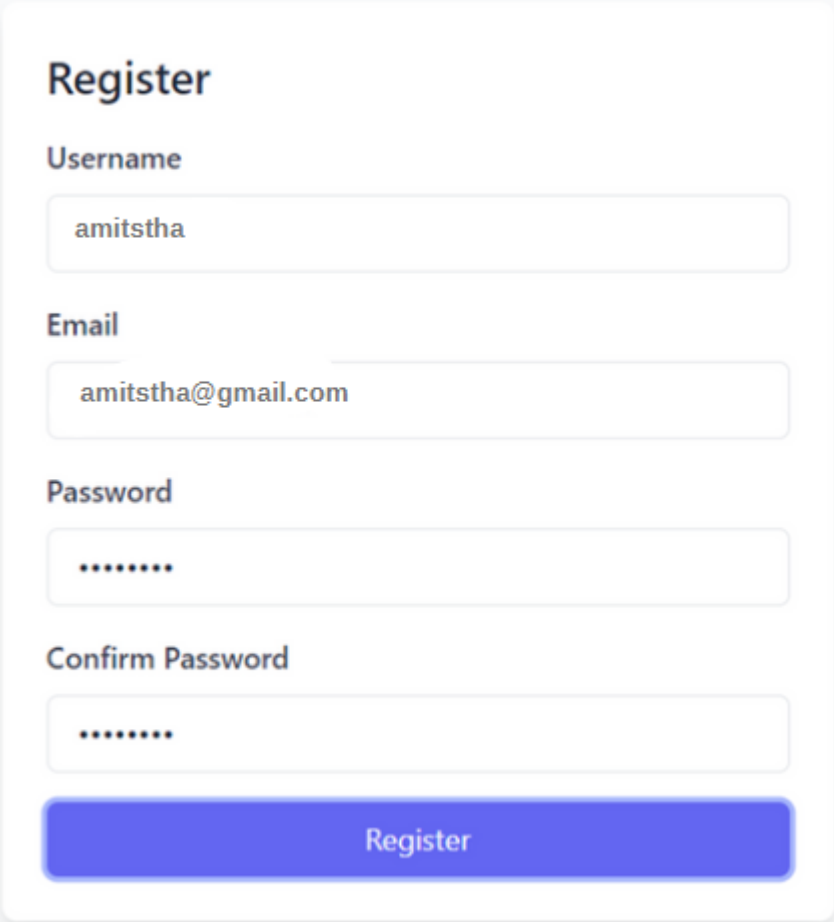
Unique Email

Design Approach

To further improve password security, the system now requires an exact match to be made between the password or email entered and the relevant information stored in the database. Passwords containing personal information are not allowed thanks to the stricter validation procedure. The approach discourages the use of readily predictable passwords and significantly reduces the risk of unauthorized access by using passwords that contain identifying information by employing a case-sensitive comparison.

Implementation

By using a unique email design process, the system makes sure that email addresses are unique when registering users. The system ensures that each email remains unique within the system by rejecting registration attempts made by users using emails that are already listed in the database with 400 error messages. This technique essentially prevents the creation of duplicate email entries, improving security and guaranteeing data integrity.



The image shows a registration form titled "Register". It contains four input fields: "Username" with the value "amitstha", "Email" with the value "amitstha@gmail.com", "Password" with masked characters "*****", and "Confirm Password" with masked characters "*****". Below the fields is a blue button labeled "Register".

Figure 3: Unique Login

```
const exist = await User.findOne({ email });
if (exist) {
  return res.status(400).json({
    success: false,
    message: "Email already exists!",
  });
}
```

TERMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

/api/v1/register 400 4.984 ms - 51

```
_id: ObjectId('65bcbf832c96190469813e0e')
username: "amit"
email: "am: i@gmail.com"
password: "$2a$10$QSPY7tZRu/csk0n8hqkRwe.1jMeJQH4xVohSVwWTP58SUU5XtLhU6"
role: "user"
createdAt: 2024-02-02T10:10:11.076+00:00
updatedAt: 2024-02-02T10:10:11.076+00:00
v: 0
```

Limited Login Attempts

Design Approach

By limiting the amount of unsuccessful login attempts in a row, the "Limit Login Attempts" feature seeks to discourage unwanted access. To prevent brute-force assaults, the system specifies the constants `MAX_ATTEMPTS_LEFT` and `DURATION_LOCKED` to monitor login attempts and verify account locks.

Implementation

```
// Check if the user has reached the maximum login attempts
if (user.loginAttempts >= 5) {
  return res.status(401).json({
    success: false,
    message: "Maximum login attempts reached. Your account is locked.",
  });
}
```

When an attempt to log in is made but fails, the system increases the value of `loginAttempts` and locks the account when the maximum number of permitted tries is reached. Then, a bad request 401 is sent, signaling that the maximum number of login attempts has been made.

Password Hashing

Design Approach

Password hashing must be used in order to ensure that user credentials are secure. The method uses bcrypt, a cryptographic hash function, to change passwords into extremely resistant and irreversible versions, decreasing the possibility of unwanted access.

Implementation

```
//hashed Password
userSchema.pre("save", async function(next){
  if(!this.isModified("password")){
    next()
  }

  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt)
})
```

Figure 4: Password Hashing

```
_id: ObjectId('65bcbf832c96190469813e0e')
username: "e--h"
email: "ams@gmail.com"
password: "$2a$10$QSPY7tZRu/csk0n8hqkRwe.1jMeJQH4xVohSVwWTP58SUU5XtLhU6"
role: "user"
createdAt: 2024-02-02T10:10:11.076+00:00
updatedAt: 2024-02-02T10:10:11.076+00:00
```

A pre-save middleware function included into the user schema encrypts passwords using bcrypt and creates a salt to boost security. The final hash is linked to the user's password field, providing strong defense against password security flaws.

Incorporation of Two-Factor Authentication (2FA)

Design Approach

The implementation of Two-Factor Authentication (2FA), which incorporates email verification, improves user account security. After creating a random verification code and linking it to the user's account, the system sends out an HTML email template with the user's code and first name on it.

Audit Logs

By providing a comprehensive log of all server requests performed for every route, the "Audit Log" feature promotes traceability, transparency, and increased security. By capturing and tracking server activity, this feature improves the capacity to identify breaches, illegal access, and guarantees regulatory compliance. It also provides an understandable and credible record of the acts carried out.

Design Approach

The Audit Log system was meticulously designed to fulfill the requirements of segregating and supervising log data associated with different routes. The implementation employed a modular methodology, using the morgan logging library for request logging, the rotating-file-stream package for log rotation, and a custom logging module to oversee the entire process. The logging architecture, showcases the systematic approach employed for security logging. Logs are arranged according to predetermined pathways in a log directory structure that the system has built. Every route has its own log folder, which guarantees that log data is separated and arranged in an orderly fashion. This approach makes it simple to manage logs, focus analysis, and rotate logs for each route in an effective manner. The logging.js module was created specifically to house route-specific logging processes. This improved code clarity allowed for versatility in the application architecture.

Implementation

The logging.js module was developed as part of the Audit Log system implementation. Its functions include log rotation and distinct log stream generation for every predefined route. This module uses the rotating-file-stream (RFS) package to ensure effective daily log rotation. The module's primary goal is to, within a base log directory, construct a log directory specific to a given route. The createLogStream(routeName) function is used to create the log directory unique to a certain route. This function generates a revolving write stream for log files related to the designated route and verifies that the necessary log directories are present.

User Access Levels

Design Approach

Web applications require access control in order to provide safe and controlled user interactions. Three levels of access are included in the design: Administrator, Registered User, and Guest User. The highest level of access is granted to administrators, who oversee user data and improve program content. Those who have registered can add products to their cart and leave comments, which helps to create a lively and interesting atmosphere. In addition to having limited access, guest users are only able to view material and cannot make purchases. They will be directed to the login page if they try to make a purchase. This highlights how crucial registration is to have complete access.

Implementation

The establishment of access control is facilitated by the backend logic and frontend interfaces. The user model has a "isAdmin" feature to distinguish between different roles. During admin registration, the "isAdmin" attribute is set to the value "true", whereas other users are assigned the value "false". After a successful login, a JSON Web Token (JWT) is created and sent to the user. This token includes the specified field. This token functions as an authentication mechanism and

provides access to specific routes based on the user's designated role. The frontend parsing procedure validates the value of the "Admin" attribute and directs the user to the suitable page. Administrators undergo a rigorous authentication process and receive authorization from backend controllers, while Registered Users are given access to functionalities. Guest users attempting to make purchases are automatically sent to the login page, with a prominent emphasis on the importance of registration.

Session Management

Design Approach

Ensuring secure user interactions requires efficient session management and expiration. After the user authenticates successfully, a special JSON Web Token (JWT) is created. This token is digitally validated with a private key and contains all of the user's data. The layout makes use of JWT's expires The ability to choose the session's duration, which could result in its automatic termination.

Implementation

```
//generateAuthToken
userSchema.methods.getJwtToken = function(){
  return jwt.sign({id:this._id},process.env.JWT_SECRET,{
    expiresIn:"1d"
  })
}
```

Figure 5: Session Management

The creation and termination of sessions are managed by the userSchema's getJwtToken method. After the authentication process is successful, a JSON Web Token (JWT) is produced. This token uses the expires In attribute to automatically end the session and stores user data. To effectively monitor session lifecycles, a session item is created in the User Session collection using the decoded JSON Web Token (JWT) data.

Feature for encrypting data

Design Approach

Vulnerable people's payment information is encrypted using the AES-256-CBC technique.

An environment variable containing an encryption key is essential to the encryption and decryption processes. A matching decode instance is used for decryption, while the AES-256-CBC cipher is used for encryption.

Implementation

Using the addLogs method is essential for putting encryption techniques into practice. Before being saved, the description field is encrypted to improve the security of private log data.

Using the supplied encryption key, the addLogs method initializes an AES-256-CBC cipher and encrypts the description input data in UTF-8 format. Hexadecimal notation is used to represent the encrypted data. The encrypted data is then effortlessly incorporated into the log entry, enhancing the logging system's dependability and security.

Conclusion

This paper investigates in detail how cutting-edge technology and robust security measures are integrated in the rapidly changing e-commerce industry. Using Next.js and the MERN stack, we have created a strong framework that combines advanced functionality with strict security controls. The main goal of our research is to create a state-of-the-art digital environment that incorporates audit logs and session management seamlessly, strengthens user authentication procedures, establishes robust access controls, and thoroughly evaluates encryption approaches. These projects highlight the crucial relationship between security and e-commerce, resulting in the creation of a strong platform able to manage the complexity of contemporary digital commerce. Users' confidence is bolstered by the extensive security procedures, which additionally safeguard critical user data. Through putting user trust and data integrity first, this study demonstrates our commitment to building a trustworthy and safe online transaction environment. E-commerce platforms must constantly adopt cutting-edge technologies and strong security procedures to maintain their development and success while keeping up with the ever-evolving digital landscape.

Link

Github : <https://github.com/amitshrestha-15/Security.git>

References

Anderson, J. (2023, April 15). The Role of Two-Factor Authentication in Enhancing Online Security. Security Today. <https://www.securitytoday.com/articles/two-factor-authentication>

Patel, M. (2022, August 12). A Comprehensive Guide to Session Management in Web Applications. Cybersecurity Insights. <https://www.cybersecurityinsights.com/session-management-guide>

Ramirez, L. (2023, July 30). Best Practices for Implementing User Access Controls. SecureTech Blog. <https://www.securetech.com/blog/user-access-controls>

Brown, T. (2024, January 10). Understanding the Importance of Data Encryption in Modern Applications. DataSec Journal. <https://www.datasecjournal.com/importance-of-data-encryption>

Walker, K. (2022, November 3). Exploring the MERN Stack: A Modern Web Development Solution. Tech Development Weekly. <https://www.techdevweekly.com/mern-stack-overview>

Lee, H. (2023, March 25). How Password Hashing Algorithms Protect User Data. Security Matters. <https://www.securitymatters.com/password-hashing-algorithms>

□Johnson, P. (2022, May 18). MongoDB: The Database of Choice for Scalable Applications. Developer's Hub. <https://www.developershub.com/mongodb-scalable-applications>

Nguyen, A. (2023, February 8). React.js: Building Dynamic User Interfaces. Web Dev Insights. <https://www.webdevinsights.com/reactjs-dynamic-interfaces>

Mitchell, S. (2023, October 14). Safeguarding Your Digital Identity: The Importance of Strong Passwords. Digital Security Blog. <https://www.digitalsecurityblog.com/strong-passwords-importance>

Adams, R. (2022, December 21). Node.js: Powering the Back-End of Modern Web Applications. CodeCraft. <https://www.codecraft.com/nodejs-backend-applications>