1. Fixing Vulnerabilities

A. Sanitize and Validate Inputs:

- **Issue Identified:** User inputs were not properly sanitized, making the application **vulnerable to XSS attacks.**
- **Action Taken:**
  - Installed validator to sanitize and validate inputs.
  - Implemented email validation and string sanitization in route handlers.
- **Code Implemented:**

```
const validator = require('validator');

// Example: Validate email input

if (!validator.isEmail(email)) {

  return res.status(400).send('Invalid email');

}

// Example: Sanitize user input (e.g., trimming spaces)

const sanitizedName = validator.trim(name);
```

B. Password Hashing:

- **Issue Identified:** Passwords were stored and transmitted in plain text, posing a risk to user data.
- **Action Taken:**
  - Installed bcrypt to hash passwords before storing them in the database.
  - Implemented password hashing during user registration and compared hashed passwords during login.
- **Code Implemented:**

```
const bcrypt = require('bcrypt');



// Hashing a password before storing it

const hashedPassword = await bcrypt.hash(password, 10);



// Verifying the password during login

const isMatch = await bcrypt.compare(inputPassword, hashedPassword);

if (!isMatch) {
  return res.status(401).send('Invalid credentials');
}
```

C. Token-Based Authentication:

- **Issue Identified:** The application lacked token-based authentication for protecting routes and managing user sessions.
- **Action Taken:**
  - Installed jsonwebtoken to implement JWT (JSON Web Token) authentication.
  - Added token generation after successful login and token verification middleware for protected routes.
- **Code Implemented:**

```
const jwt = require('jsonwebtoken');



// Generate JWT token after successful login

const token = jwt.sign({ id: user._id }, 'your-secret-key', { expiresIn: '1h' });



// Send the token to the client

res.send({ token });
```

D. Secure Data Transmission:

- **Issue Identified:** Sensitive data was transmitted without proper encryption, and security headers were not set.
- **Action Taken**
  - Installed helmet to set various HTTP headers to secure the application.
  - Added helmet middleware to secure HTTP headers and prevent certain types of attacks.
- **Code Implemented:**

```
const helmet = require('helmet');

app.use(helmet());
```

2. Summary of Actions Taken:

- **Sanitized and Validated User Inputs:**
  Implemented input validation using the validator library to prevent XSS attacks and ensure data integrity.
- **Implemented Password Hashing:**
  Used bcrypt to hash passwords, ensuring sensitive data is stored securely.
- **Enhanced Authentication with JWT:**
  Added token-based authentication using jsonwebtoken to protect routes and manage

user sessions.

- **Secured Data Transmission:**
  Implemented helmet to secure HTTP headers and prevent common attacks.

3. Vulnerabilities Addressed:

- **XSS (Cross-Site Scripting):** Prevented by sanitizing and validating user inputs.
- **Weak Password Storage:** Mitigated by hashing passwords using bcrypt.
- **Lack of Authentication:** Addressed by adding JWT token-based authentication.
- **Insecure Data Transmission:** Secured using HTTPS and helmet to set secure HTTP headers.