Bug Report Document

Overview

This document lists and details the bugs identified in the provided Flask application code for user registration and login functionalities. Each issue is analyzed with clear steps to reproduce, expected and actual results, severity, priority, and suggested fixes.

Bug Details

1. Missing Input Validation

- **ID**: BUG001
- **Summary**: No input validation implemented for form fields.
- **Steps to Reproduce**:
    1. Navigate to the `/client_registeration` endpoint.
    2. Enter invalid email addresses, such as `"joan@",` and even only spaces `"  "`.
    3. Submit the form.
- **Expected Result**: The system should validate email inputs and reject invalid formats.
- **Actual Result**: The system accepts invalid email formats with leading/trailing spaces.
- **Severity**: Critical
- **Priority**: High
- **Suggested Fix**: Implement server-side validation for all fields, especially email.

2. Password Not Hashed

- **ID**: BUG002
- **Summary**: Passwords are stored in plain text, which is a significant security vulnerability.
- **Steps to Reproduce**:
    1. Register a user via the `/client_registeration` endpoint.
    2. Check the database for stored user data.
- **Expected Result**: Passwords should be hashed using secure algorithms.
- **Actual Result**: Passwords are stored in plain text in the database.
- **Severity**: Critical
- **Priority**: High
- **Suggested Fix**: Use a password-hashing library to hash and securely store passwords.

3. Inadequate Error Messages

- **ID**: BUG003
- **Summary**: Error messages do not specify the field or issue causing validation failure.
- **Steps to Reproduce**:
    1. Leave one or more fields empty on the registration form.
    2. Submit the form.

- **Expected Result**: Error messages should specify which field(s) are invalid.
- **Actual Result**: Generic message `{"msg": "Invalid Data"}` is returned.
- **Severity**: Moderate
- **Priority**: Medium
- **Suggested Fix**: Return detailed error messages for each invalid field.

4. Token Logging

- **ID**: BUG004
- **Summary**: Sensitive JWT tokens are logged, which poses a security risk.
- **Steps to Reproduce**:
    1. Login using the `/client_login` endpoint.
    2. Review server logs.
- **Expected Result**: JWT tokens should not appear in server logs.
- **Actual Result**: Tokens are logged, potentially exposing sensitive user data.
- **Severity**: High
- **Priority**: High
- **Suggested Fix**: Remove logging of sensitive data, especially JWT tokens.

5. Hardcoded Secrets

- **ID**: BUG005
- **Summary**: Secret keys are hardcoded in the application code.
- **Steps to Reproduce**:
    1. Review the Flask application code.
    2. Locate the secret keys.
- **Expected Result**: Secrets should be managed using a secure service (e.g., AWS Secrets Manager, environment variables).
- **Actual Result**: Secrets are hardcoded, making the application vulnerable to unauthorized access.
- **Severity**: Critical
- **Priority**: High
- **Suggested Fix**: Implement secure secret management practices.

6. No CSRF Protection

- **ID**: BUG006
- **Summary**: CSRF protection is missing, making the application vulnerable to cross-site request forgery attacks.
- **Expected Result**: CSRF tokens should be implemented and validated for all POST requests.
- **Actual Result**: No CSRF protection is present.
- **Severity**: High
- **Priority**: High
- **Suggested Fix**: Use CSRF protection libraries like Flask-WTF.

7. No Rate Limiting

- **ID**: BUG007
- **Summary**: The application does not implement rate limiting, leaving it vulnerable to brute-force and DDoS attacks.
- **Steps to Reproduce**:
    1. Continuously send login requests to the `/client_login` endpoint.
    2. Observe the system's behavior.
- **Expected Result**: Requests should be rate-limited to prevent abuse.
- **Actual Result**: Unlimited requests are accepted without restriction.
- **Severity**: High
- **Priority**: High
- **Suggested Fix**: Implement rate limiting using Flask-Limiter or similar libraries.

8. JWT Token Expiry Not Set

- **ID**: BUG008
- **Summary**: JWT tokens lack an expiration time, potentially compromising security.
- **Steps to Reproduce**:
    1. Log in via the `/client_login` endpoint.
    2. Decode the JWT token.
- **Expected Result**: Tokens should have an expiration time to limit their validity.
- **Actual Result**: Tokens remain valid indefinitely.
- **Severity**: High
- **Priority**: Medium
- **Suggested Fix**: Add an expiration time to JWT tokens and enforce its validation.

9. Only Form data is acceptable

- **ID**: BUG000
- **Summary**: API handles only form data but no support for JSON request payload.
- **Steps to Reproduce**:
    1. Send request to any endpoint using JSON payload content type
- **Expected Result**: API should ideally handle JSON request payload .
- **Actual Result**: API only handles form data.
- **Severity**: Medium
- **Priority**: Medium
- **Suggested Fix**: Modify API such that it accepts JSON payload.