Bug Report for Logical Testing

Bug Report for `/client_registeration`

Logical_Issue-1

Title:

Invalid Email Format Accepted in `/client_registeration` Endpoint

Priority:

Medium

Description:

The `/client_registeration` endpoint accepts invalid email formats during user registration. This allows incorrect data to be stored in the database, leading to potential issues with user management and email-based operations.

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `userName`: `username`
        - `email`: `abcd`
        - `password`: `password123`
        - `phone`: `1234567890`
3. Observe the response:

- Response: `200 OK`
- Message: `{"msg": "User Registered"}`

Expected Result:

The system should validate the `email` field and reject inputs that do not match a valid email format (e.g., `user@example.com`).

Actual Result:

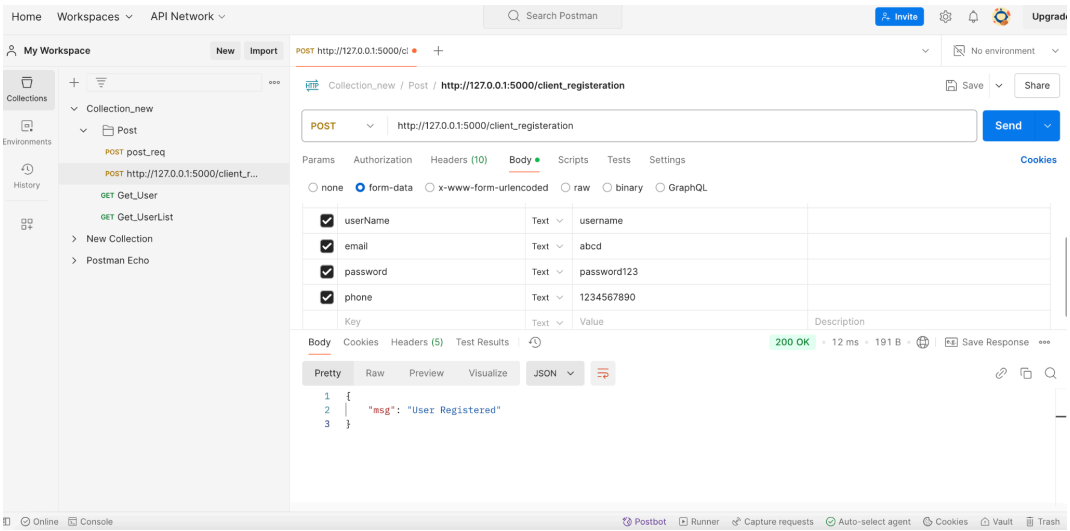
The system accepts invalid email formats and registers the user successfully, storing the invalid data in the database.

Impact:

1. **Data Integrity:** Storing invalid email addresses can lead to issues in user communication and management.
2. **Business Impact:** Email-based operations, such as account verification and password recovery, may fail.
3. **User Experience:** Users may face issues when attempting to log in or perform actions tied to their email address.

Attachments:

1. Screenshot of the API request and response showing successful registration with an invalid email format:



Environment:

- **Browser:** Postman (Version 11.22.1)
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** Medium (5/10)
- **Impact:** Medium (6/10)
- **Overall Risk Score:** 5.5/10 (Medium)

Logical_Issue-2

Title:

Weak Password Accepted in `/client_registeration` Endpoint

Priority:

High

Description:

The `/client_registeration` endpoint allows weak passwords (e.g., short or common passwords) to be registered, compromising the security of user accounts. The system does not enforce any password strength policy.

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `userName`: `username`
        - `email`: `user@gmail.com`
        - `password`: `12345`
        - `phone`: `1234567890`
3. Observe the response:
    - Response: `200 OK`
    - Message: `{"msg": "User Registered"}`

Expected Result:

The system should enforce a password strength policy, rejecting passwords that are:

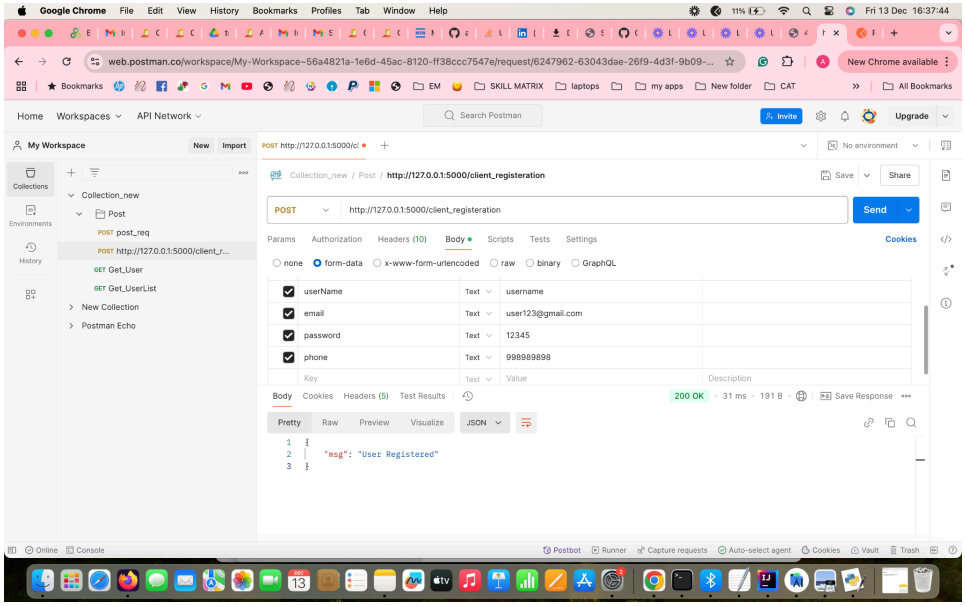
- Too short (e.g., less than 8 characters).
- Common or predictable (e.g., `12345`, `password`).

- Lacking complexity (e.g., no mix of uppercase letters, numbers, and symbols).

Actual Result:

The system accepts weak passwords and registers the user successfully, storing the weak password in the database.

Attachments:



Environment:

- **Browser:** Postman (Version 11.22.1)
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** High (8/10)
- **Impact:** High (8/10)
- **Overall Risk Score:** 8/10 (High)

Logical_Issue-3

Title:

Invalid Phone Number Accepted in `/client_registeration` Endpoint

Priority:

Medium

Description:

The `/client_registeration` endpoint does not validate phone numbers correctly and allows invalid phone number formats (e.g., alphanumeric or containing special characters). This can lead to data integrity issues and failures in operations dependent on valid phone numbers.

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `fullName`: `Username Test`
        - `userName`: `Username`
        - `email`: `tst@gmail.com`
        - `password`: `password123`
        - `phone`: `123456789068686@@@@&&`
3. Observe the response:
    - Response: `200 OK`
    - Message: `{"msg": "User Registered"}`

Expected Result:

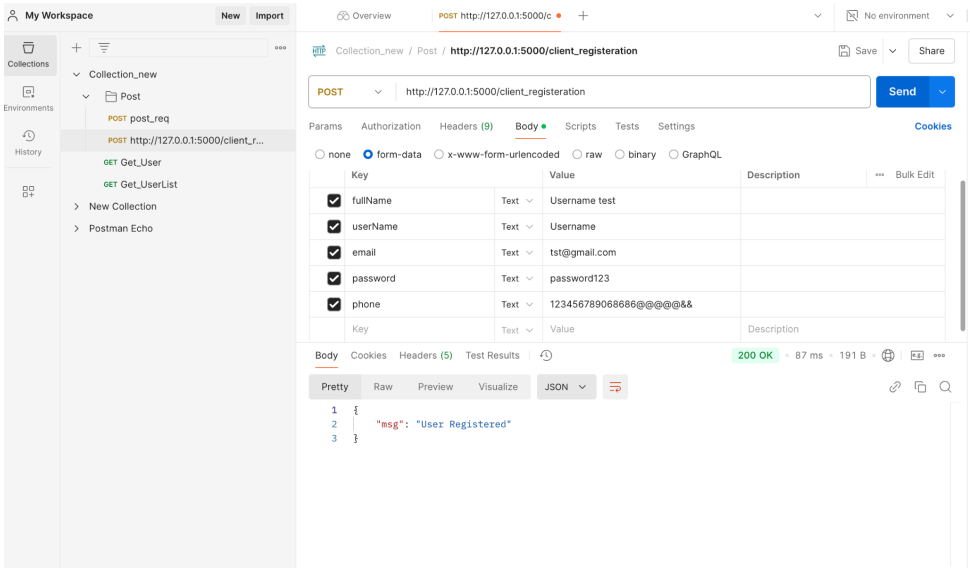
The system should validate the `phone` field and reject inputs that:

- Contain non-numeric characters.
- Are not in a valid phone number format (e.g., country code followed by 10 digits).

Actual Result:

The system accepts invalid phone numbers and registers the user successfully, storing the invalid data in the database.

Attachments:



Environment:

- **Browser:** Postman (Version 11.22.1)
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** Medium (6/10)
- **Impact:** Medium (6/10)
- **Overall Risk Score:** 6/10 (Medium)

Logical_Issue-4

Title:

Duplicate Username Allowed in `/client_registeration` Endpoint

Priority:

High

Description:

The `/client_registeration` endpoint does not validate the uniqueness of the `userName` field, allowing multiple users to register with the same username. This can lead to conflicts in user identification and functionality dependent on the `userName` field.

Steps to Reproduce:

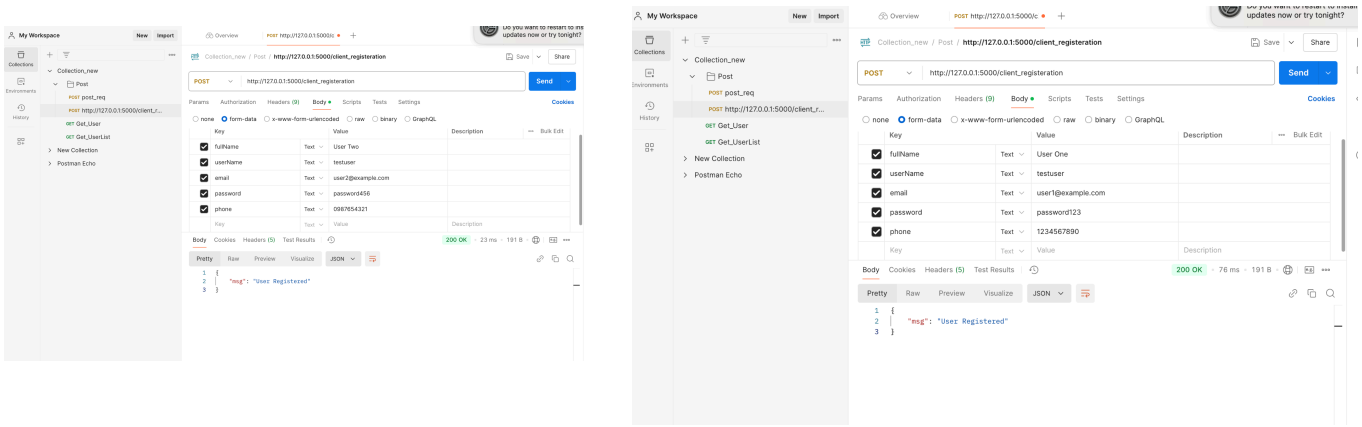
1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **Request 1:**
        - **form-data**:
            - `fullName`: User One
            - `userName`: testuser
            - `email`: `user1@example.com`
            - `password`: `password123`
            - `phone`: `1234567890`
        - Response: `200 OK`, `{"msg": "User Registered"}`
    - **Request 2:**
        - **form-data**:
            - `fullName`: `User Two`
            - `userName`: testuser
            - `email`: `user2@example.com`
            - `password`: `password456`
            - `phone`: `0987654321`
        - Response: `200 OK`, `{"msg": "User Registered"}`

Expected Result:

The system should validate the `userName` field for uniqueness and reject duplicate usernames with an appropriate error message (e.g., `"Username already exists"`). Duplicate usernames can cause conflicts in identifying users and mapping their data.

Actual Result:

The system allows multiple users to register with the same `userName` and returns a success response for both registrations.



Attachments:

Environment:

- **Browser:** Postman (Version 11.22.1)
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** High (8/10)
- **Impact:** High (8/10)
- **Overall Risk Score:** 8/10 (High)

Logical_Issue-5

Title:

Privilege Hardcoding in `/client_registeration` Endpoint

Priority:

Medium

Description:

The `privillage` (privilege) field is hardcoded to `2` for all users during registration. This approach lacks flexibility to handle different privilege levels, such as admin users, moderators, or regular users. It limits the scalability and functionality of the system in environments where role-based access control (RBAC) is required.

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `fullName`: `Test User`
        - `userName`: `testuser`
        - `email`: `testuser@example.com`
        - `password`: `password123`
        - `phone`: `1234567890`
3. Observe the database after the registration.
    - The `privillage` column for all users is set to `2`.

Expected Result:

The `privillage` field should:

1. Be determined dynamically based on the role assigned to the user (e.g., `1` for admin, `2` for regular user).

2. Optionally accept a value from the registration payload or be derived from business logic.

Actual Result:

The system hardcodes the `privillage` field as `2` for all users, regardless of their role or access level.

Environment:

- **Postman Version:** 11.22.1
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** Medium (5/10)
- **Impact:** Medium (6/10)
- **Overall Risk Score:** 5.5/10 (Medium)

Logical_Issue-6

Title:

Case Sensitivity in Duplicate Email Check Allows Duplicate Registrations

Priority:

Medium

Description:

The `/client_registeration` endpoint does not handle case insensitivity for email addresses during the duplicate email check. As a result, the system allows multiple registrations with email addresses that differ only by case (e.g., `User@Example.com` and `user@example.com`).

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **Request 1:**
        - **form-data**:
            - `fullName`: `User One`
            - `userName`: `testuser1`
            - `email`: `User@Example.com`
            - `password`: `password456`
            - `phone`: `0987654321`
        - Response: `200 OK`, `{"msg": "User Registered"}`
    - **Request 2:**
        - **form-data**:
            - `fullName`: `User Two`
            - `userName`: `testuser2`
            - `email`: `user@example.com`
            - `password`: `password456`
            - `phone`: `0987654321`
        - Response: `200 OK`, `{"msg": "User Registered"}`
3. Check the database to confirm that both emails are registered as separate users.

Expected Result:

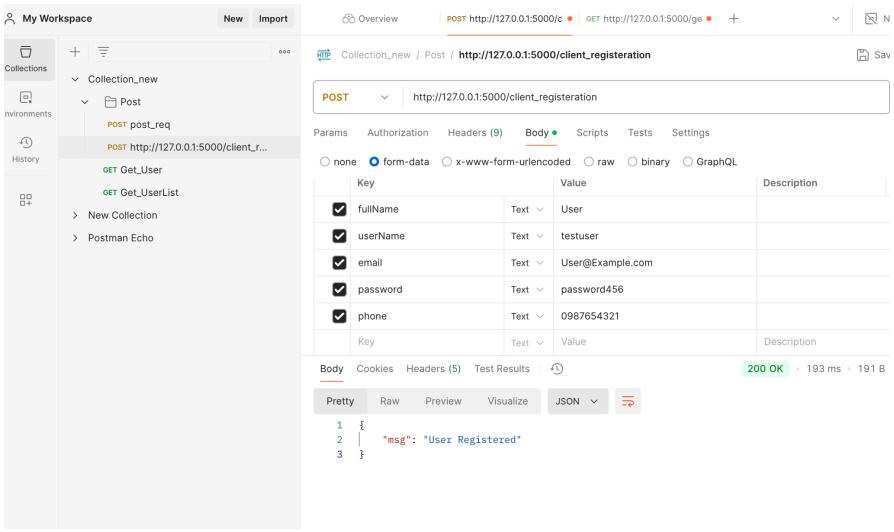
The system should treat email addresses as case-insensitive and reject duplicate email registrations regardless of case.

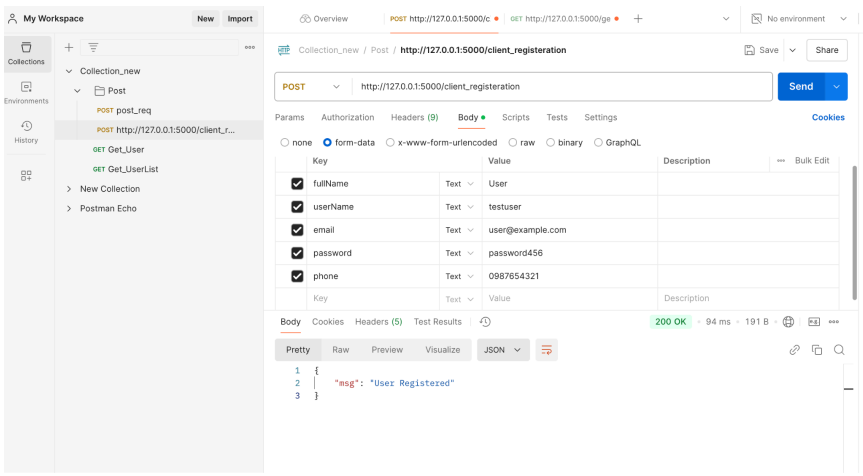
Actual Result:

The system allows duplicate email registrations where the only difference is the casing of the email address.

Attachments:

1. Screenshot of the API requests and responses showing successful registration with duplicate emails (case-sensitive).





Environment:

- **Postman Version:** 11.22.1
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** Medium (5/10)
- **Impact:** Medium (6/10)
- **Overall Risk Score:** 5.5/10 (Medium)

Bug Report for Security Testing in `/client_registeration`

Security_Issue-1

Title:

SQL Injection Vulnerability in `/client_registeration` Endpoint

Priority:

Critical

Description:

A SQL Injection vulnerability exists in the `/client_registeration` endpoint of the Flask-based application. This issue allows attackers to manipulate the SQL query by injecting malicious input into the `email` parameter, potentially bypassing validation and compromising database integrity.

Steps to Reproduce:

1. Start the Flask application locally.
2. Open **Postman** or any API testing tool.
3. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `fullName`: `Username test`
        - `userName`: `Username`
        - `email`: `user@example.com OR 1=1`
        - `password`: `password123`
        - `phone`: `1234567890`
4. Observe the response:
    - Response: `200 OK`
    - Message: `{"msg": "User Registered"}`

Check the database. The SQL query executed is:
SELECT userName FROM users WHERE email = "user@example.com" OR 1=1

5. This query can fetch all rows from the `users` table due to the injected condition `1=1`.

Expected Result:

The application should validate and sanitize user input, rejecting malicious payloads. It should respond with an appropriate error message, such as `"Invalid email"`.

Actual Result:

The application accepts the malicious payload, successfully registers the user, and returns a `200 OK` status with the message `User Registered`.

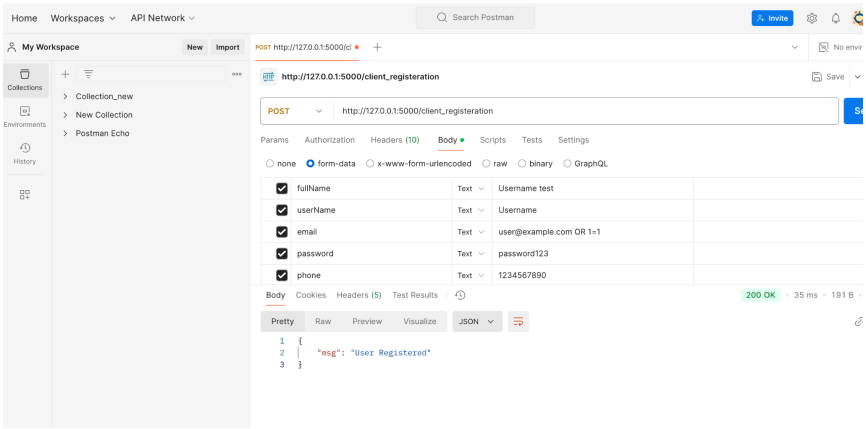
Impact:

This vulnerability poses a critical risk:

1. **Database Exposure:** An attacker can access sensitive data stored in the database.
2. **Authentication Bypass:** By injecting conditions like `OR 1=1`, attackers can bypass email checks or access restricted functionalities.
3. **Potential Data Loss:** Further exploitation could lead to unauthorized data manipulation or deletion.

Attachments:

1. Screenshot of the API request in Postman.



Environment:

- **Browser:** Postman (11.22.1)
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** High (8/10)
- **Impact:** High (9/10)
- **Overall Risk Score:** 9/10 (Critical)

Security_Issue-2

Title:

Cross-Site Scripting (XSS) Vulnerability in `/client_registeration` Endpoint

Priority:

High

Description:

The `/client_registeration` endpoint does not sanitize user inputs properly, allowing attackers to inject malicious scripts into fields such as `fullName`. This vulnerability can lead to XSS attacks, which could compromise the security and integrity of the application.

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `fullName`: `<script>alert('XSS')</script>`
        - `userName`: `testuser`
        - `email`: `user123@example.com`
        - `password`: `password456`
        - `phone`: `0987654321`
3. Observe the response:
    - Response: `200 OK`, `{"msg": "User Registered"}`
4. If the `fullName` field is rendered on the UI without sanitization, the malicious script executes.

Expected Result:

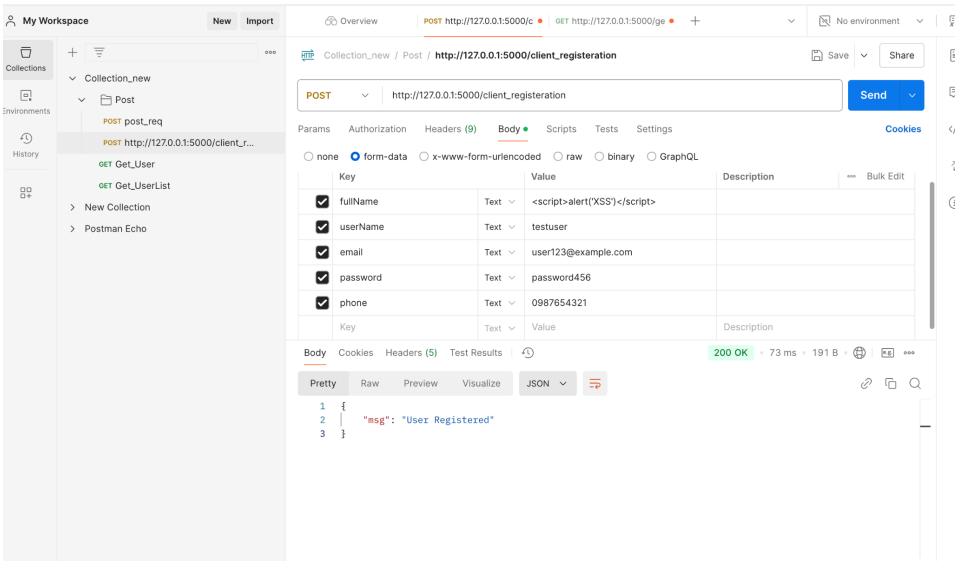
The application should sanitize all user inputs and reject scripts or any malicious payload. The `fullName` field should not allow special characters or HTML/JavaScript tags.

Actual Result:

The application accepts the malicious input without sanitization, potentially allowing the script to execute in contexts where the `fullName` field is displayed (e.g., user profiles, admin dashboards).

Attachments:

1. Screenshot of the API request and response showing successful registration with XSS



payload:

Environment:

- **Postman Version:** 11.22.1
- **OS:** macOS (Sonoma 14.0 (23A344))

- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** High (8/10)
- **Impact:** High (9/10)
- **Overall Risk Score:** 8.5/10 (Critical)

Security_Issue-3

Title:

Privilege Escalation via Parameter Manipulation in `/client_registeration` Endpoint

---

Priority:

High

Description:

The `/client_registeration` endpoint allows unauthorized privilege escalation through parameter manipulation. Attackers can add a `privillage` field in the request payload to assign themselves elevated privileges (e.g., `admin`), bypassing server-side validation.

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `fullName`: `user`
        - `userName`: `testuser`
        - `email`: `user111@example.com`
        - `password`: `password456`
        - `phone`: `0987654321`
        - `privillage`: `1` (manually added field)
3. Observe the response:
    - Response: `200 OK`, `{"msg": "User Registered"}`

Expected Result:

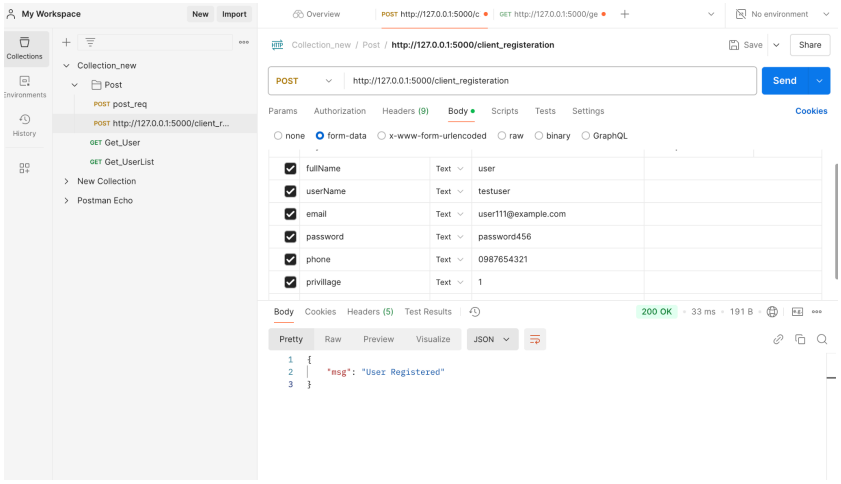
The server should ignore unauthorized parameters like `privillage` or validate their values against a whitelist of acceptable inputs based on user roles.

Actual Result:

The server accepts the `privillage` parameter directly from the payload without validation, allowing attackers to escalate their privileges.

Attachments:

1. Screenshot of the API request and response showing successful privilege escalation:



Environment:

- **Postman Version:** 11.22.1
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** High (8/10)
- **Impact:** Critical (9/10)
- **Overall Risk Score:** 8.5/10 (Critical)

Security_Issue-4

Title:

Parameter Tampering Vulnerability in `/client_registeration` Endpoint

Priority:

High

Description:

The `/client_registeration` endpoint is vulnerable to parameter tampering. Attackers can inject malicious SQL commands and invalid data types into the input fields, such as `email` and `phone`. The server processes these inputs without validation, which could result in SQL injection, database corruption, or broken functionality.

Steps to Reproduce:

1. Open **Postman** or any API testing tool.
2. Send a `POST` request to `http://127.0.0.1:5000/client_registeration` with the following payload:
    - **form-data**:
        - `fullName`: `user`
        - `userName`: `testuser`
        - `email`: `UNION SELECT * FROM users; --`
        - `password`: `1234567`
        - `phone`: `abc123!@#`
3. Observe the response:
    - Response: `200 OK`, `{"msg": "User Registered"}`
4. Verify the database or application behavior. The SQL query could fetch unintended data, and invalid `phone` data could cause inconsistencies.

Expected Result:

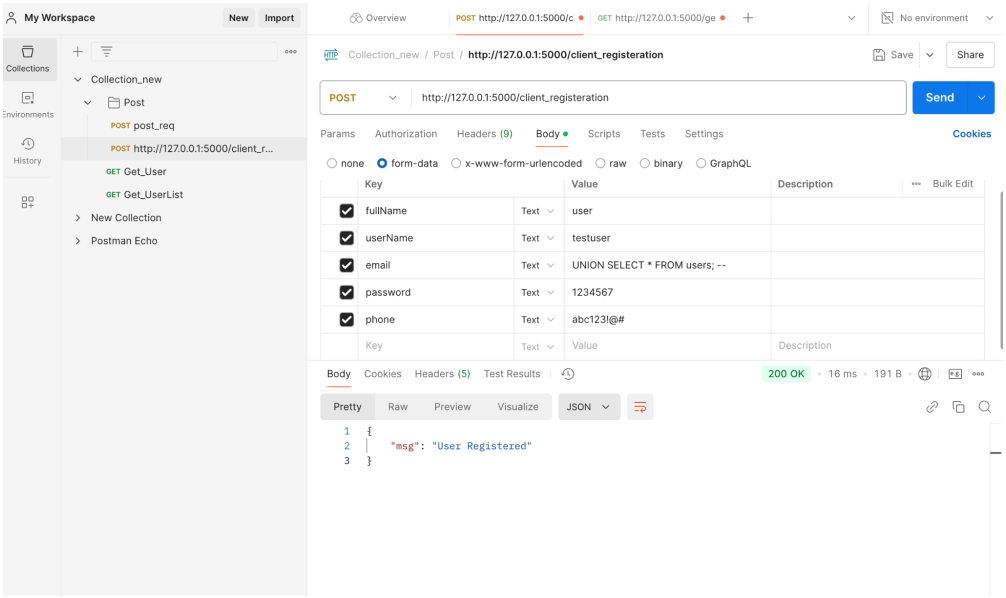
The server should validate all input fields and reject malicious payloads or invalid data types. The `email` and `phone` fields should be sanitized and validated for proper format.

Actual Result:

The server accepts malicious SQL commands and invalid data without validation, potentially executing the query or corrupting the database.

Attachments:

1. Screenshot of the API request and response showing successful registration with tampered parameters:



Environment:

- **Postman Version:** 11.22.1
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** High (8/10)
- **Impact:** High (9/10)
- **Overall Risk Score:** 8.5/10 (Critical)

Security_Issue-5

Title:

Insecure Communications via HTTP in `/client_registeration` Endpoint

Priority:

Critical

Description:

The `/client_registeration` endpoint uses plain HTTP instead of HTTPS for communication. This exposes sensitive user data, such as passwords and personally identifiable information (PII), to potential interception by attackers. HTTPS is essential to secure data transmission and protect users from man-in-the-middle (MITM) attacks.

Steps to Reproduce:

Access the `/client_registeration` endpoint:
POST http://127.0.0.1:5000/client_registeration

1.
2. Submit a registration request with sensitive data in the payload:
    - **form-data**:
        - `fullName`: `Test User`
        - `userName`: `testuser`
        - `email`: `testuser@example.com`
        - `password`: `password123`
        - `phone`: `1234567890`
3. Observe that the communication occurs over HTTP,

Expected Result:

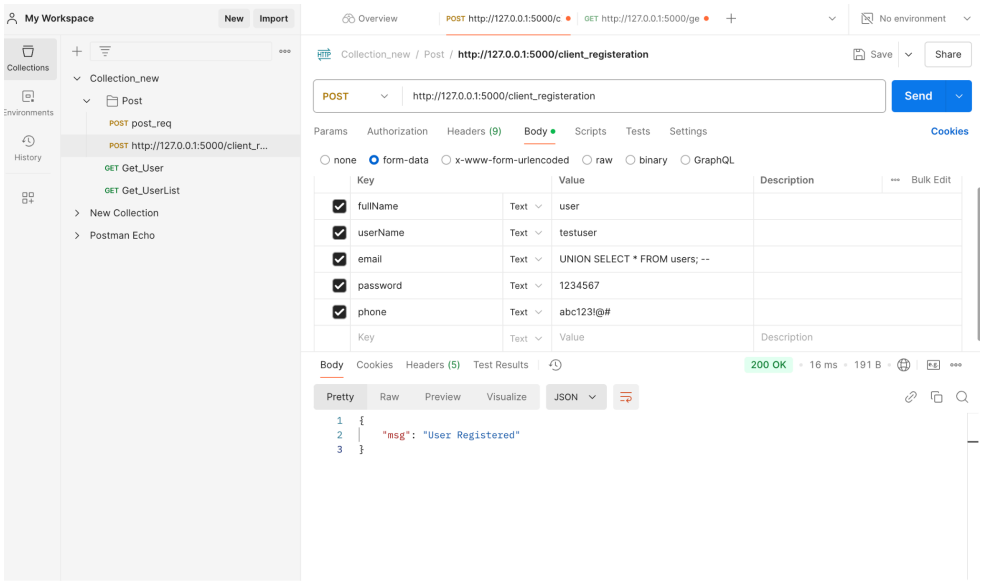
The application should enforce HTTPS for all communications to ensure data is encrypted during transmission.

Actual Result:

The `/client_registeration` endpoint accepts requests over HTTP, transmitting data in plaintext.

Attachments:

1. Screenshot of the Postman request showing communication over HTTP.



Environment:

- **Postman Version:** 11.22.1
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost: 127.0.0.1:5000)

Risk Scoring:

- **Likelihood:** High (9/10)
- **Impact:** Critical (10/10)
- **Overall Risk Score:** 9.5/10 (Critical)

Security_Issue-6

Title:

CSRF Vulnerability Allows Malicious Forms to Be Added and Executed via Console

---

Priority:

High

Description:

The `/client_registeration` endpoint is vulnerable to Cross-Site Request Forgery (CSRF). Using the browser console, attackers can add and submit unauthorized forms to manipulate the system or register users without consent. The endpoint lacks proper CSRF protection mechanisms to prevent such attacks.

Steps to Reproduce:

1. Go to the application URL: `http://127.0.0.1:5000/`.
2. Open the browser's developer console.

Enter the following script to add a malicious form:

```
document.body.innerHTML += `
  <form action="http://127.0.0.1:5000/client_registeration" method="POST">
    <input type="hidden" name="email" value="malicious@example.com">
    <input type="hidden" name="password" value="password123">
    <input type="submit">
  </form>
`;
console.log("Form added successfully.");
```

3.
4. Observe that the form is added successfully to the page, and users could unknowingly submit it.

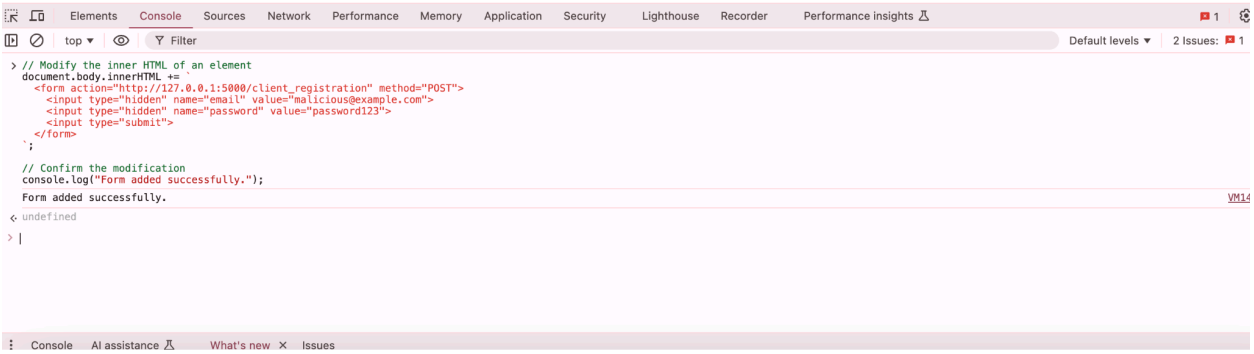
Expected Result:

The server should implement CSRF protection to prevent unauthorized forms from being added or submitted. The form submission should fail without a valid CSRF token.

Actual Result:

The server allows the form to be added and processed without validating the origin of the request or requiring a CSRF token.

Attachments:



Environment:

- **URL:** `http://127.0.0.1:5000/`
- **Browser:** Chrome
- **OS:** macOS (Sonoma 14.0 (23A344))
- **Backend:** Flask Application (Localhost)

Risk Scoring:

- **Likelihood:** High (8/10)
- **Impact:** High (9/10)
- **Overall Risk Score:** 8.5/10 (Critical)