Week-01 Task:

Submitted By:

Iqra Azam

1. Overview

The objective of this task was to analyze a simple User Management System for basic security vulnerabilities. The goal was to understand how common vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection, and security misconfigurations can be identified using basic testing techniques and browser tools.

2. Application Used

For this task, a mock web-based User Management application was used from GitHub.

GitHub Repository Link:
https://github.com/NipunChamika/reactjs_user_management

This application includes basic features such as:

- User Login
- User Registration (Signup)
- Profile-related input forms

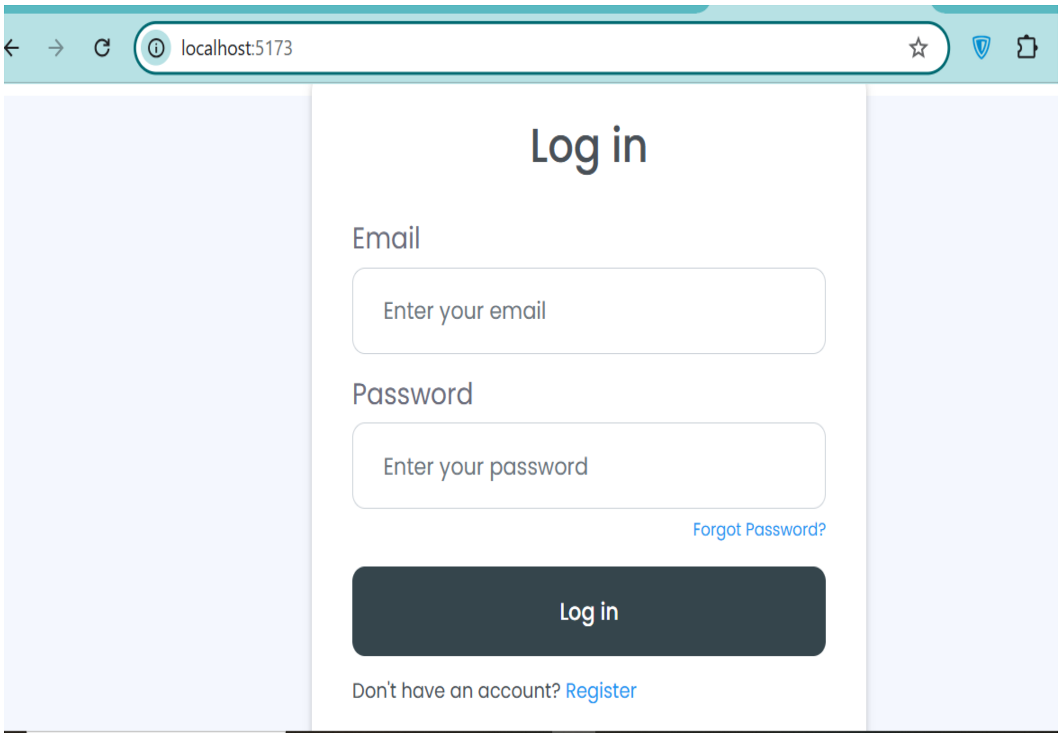
The application was used only for **security testing and learning purposes**.

3. Application Setup

The following steps were performed to run the application locally:

1. The repository was cloned from GitHub.
2. The project folder was opened in the terminal.
3. Dependencies were installed using:
4. `npm install`
5. The application was started using:

6. `npm run dev`
7. The application was opened in the browser at:
8. `http://localhost:5173`



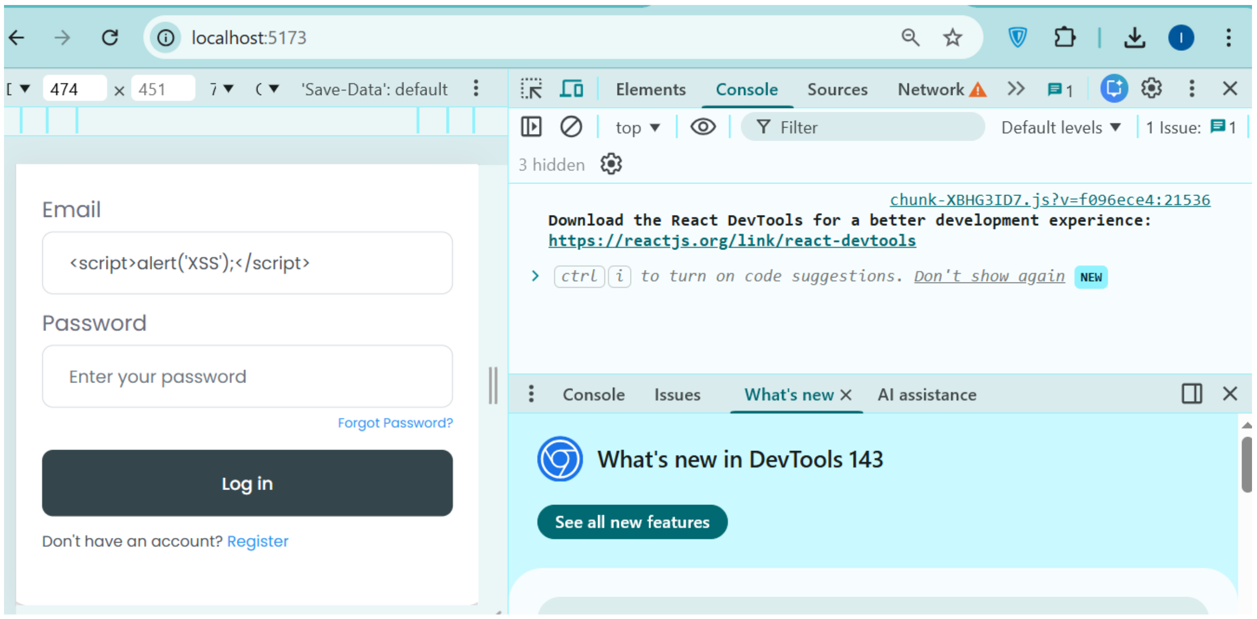
The login and registration pages were accessible successfully.

4. Security Testing Performed

4.1 Cross-Site Scripting (XSS) Testing

Steps Performed:

1. The Register page was opened.
2. In the email input field, the following script was entered:
3. `<script>alert('XSS');</script>`
4. Browser Developer Tools were opened using **Right Click → Inspect → Console**.



```
chunk-XBHG3ID7.js?v=f096ece4:21536
Download the React DevTools for a better development experience:
https://reactjs.org/link/react-devtools
```

Observation:

- The application does not properly sanitize user input.
- No backend execution occurred, but the input was accepted on the client side.

Finding:

The application is vulnerable to potential Cross-Site Scripting (XSS) attacks due to lack of proper input sanitization.

4.2 SQL Injection Testing

Steps Performed:

1. The Login page was opened.
2. The following payload was entered:
    - Email:
    - `admin@test.com' OR '1'='1`
    - Password:
    - `admin' OR '1'='1`
3. The Login button was clicked.

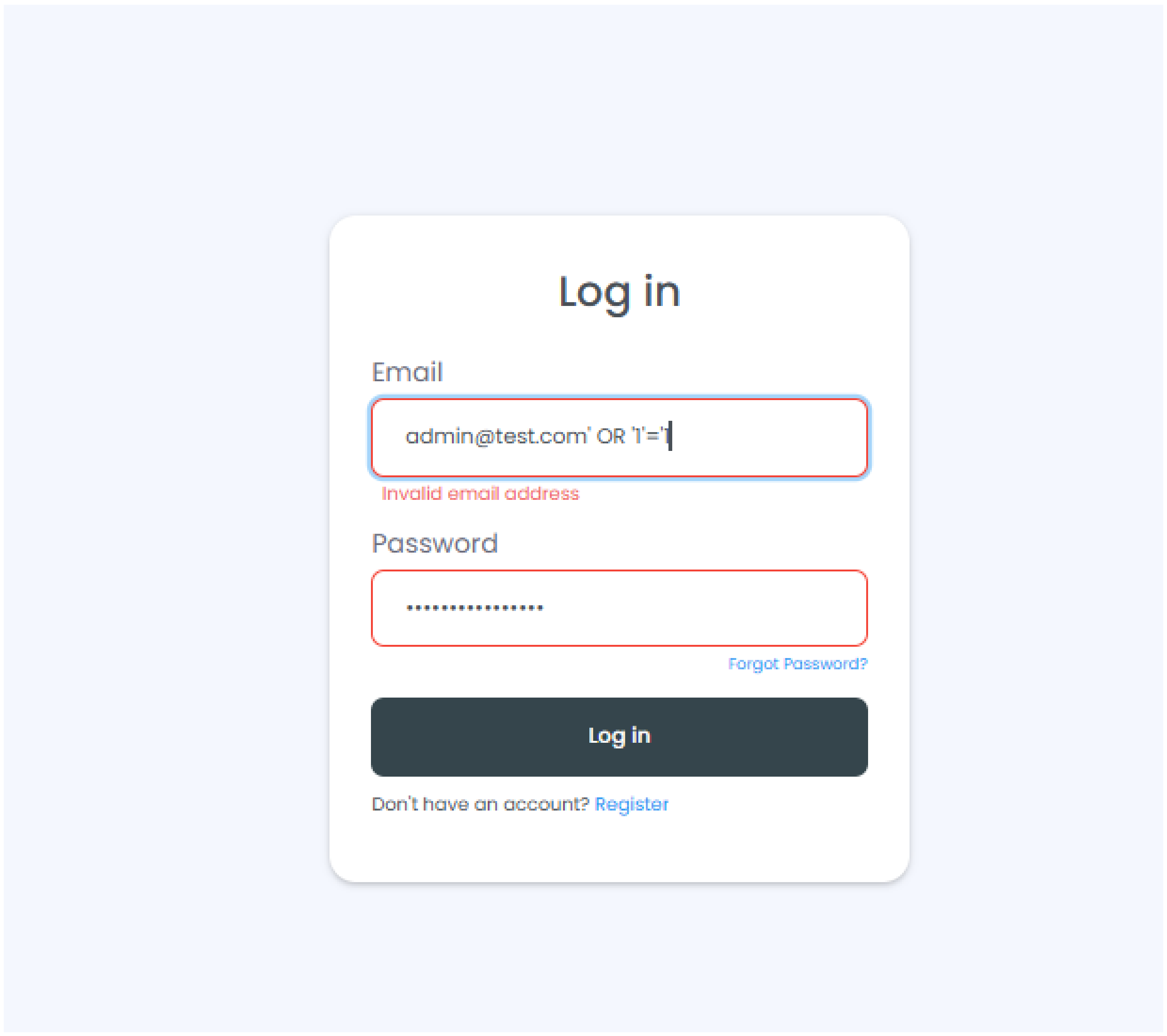
Observation:

- The application displayed an **"Invalid email address"** error.

- This indicates client-side email format validation is present.
- Backend validation could not be verified because the server was not connected.

Finding:

SQL Injection was tested on the login form. Client-side validation prevents malformed email input; however, backend-side validation could not be verified.



Log in

Email

admin@test.com' OR '1'='1

Invalid email address

Password

••••••••••••••••

4.3 Backend & Security Misconfiguration

Observation:

- During signup and login attempts, the following error appeared in the browser console:
- `AxiosError: ERR_CONNECTION_REFUSED`
- This indicates that the backend server is not running or not properly configured.

Finding:

The application shows backend connectivity issues, indicating a security misconfiguration in the mock setup.

5. Vulnerabilities Identified

- Potential Cross-Site Scripting (XSS)
- Input fields lack proper sanitization
- Backend server not connected (security misconfiguration)
- Limited backend validation visibility
- No visible Content Security Policy (CSP)

6. Areas of Improvement

- Sanitize and validate all user inputs on both client and server side
- Implement proper backend server configuration
- Use password hashing techniques (e.g., bcrypt)
- Apply Content Security Policy (CSP)
- Improve error handling and security headers

7. Conclusion

This task helped in understanding how basic security testing is performed on web applications. Through simple tools such as browser developer tools and manual payload testing, common vulnerabilities can be identified even in mock applications. This exercise improved awareness of web security fundamentals and secure development practices.