Information Security Project:

Web Application Vulnerability Assessment and Mitigation



Session 2023-27

**Submitted by:**

Amir Hashmi 2023-CS-11

Sher Muhammad 2023-CS-15

Mobeen Butt 2023-CS-28

**Supervised by:**

Dr. Faiza Iqbal

**Course:**Information Security

**Department of Computer Science**

**University of Engineering and Technology Lahore Pakistan**

**Table of Contents**

**[1. Introduction 2](#_Toc196595820)**

**[2. Vulnerabilities Exploited & Mitigation Strategies 2](#_Toc196595821)**

**[2.1.1 Distributed Denial of Service (DDoS) Attack 2](#_Toc196595822)**

**[2.1.2 Cross-Site Scripting (XSS) Attack 3](#_Toc196595823)**

**[2.1.3. Cross-Site Request Forgery (CSRF) Protection 6](#_Toc196595824)**

**[2.1.4. Brute Force Password Attack 8](#_Toc196595825)**

**[2.1.5. SQL Injection Attack 10](#_Toc196595826)**

**[2.1.6. Location-Based Access Restriction 10](#_Toc196595827)**

**[2.1.7. Password Encryption (Feistel Cipher) 12](#_Toc196595828)**

**[3. Conclusion: 13](#_Toc196595829)**

**[4. References: 13](#_Toc196595830)**

## ****1. Introduction****

This project focuses on identifying, exploiting, and mitigating common security vulnerabilities in a self-developed website. The goal was to simulate real-world attacks, implement security measures, and verify their effectiveness.

**Objectives:**

* Identify vulnerabilities in a web application.
* Perform attacks to exploit these vulnerabilities.
* Implement security solutions to mitigate risks.
* Re-test the application to ensure vulnerabilities are patched.

## ****2. Vulnerabilities Exploited & Mitigation Strategies****

### ****2.1.1 Distributed Denial of Service (DDoS) Attack****

**Attack Description:**

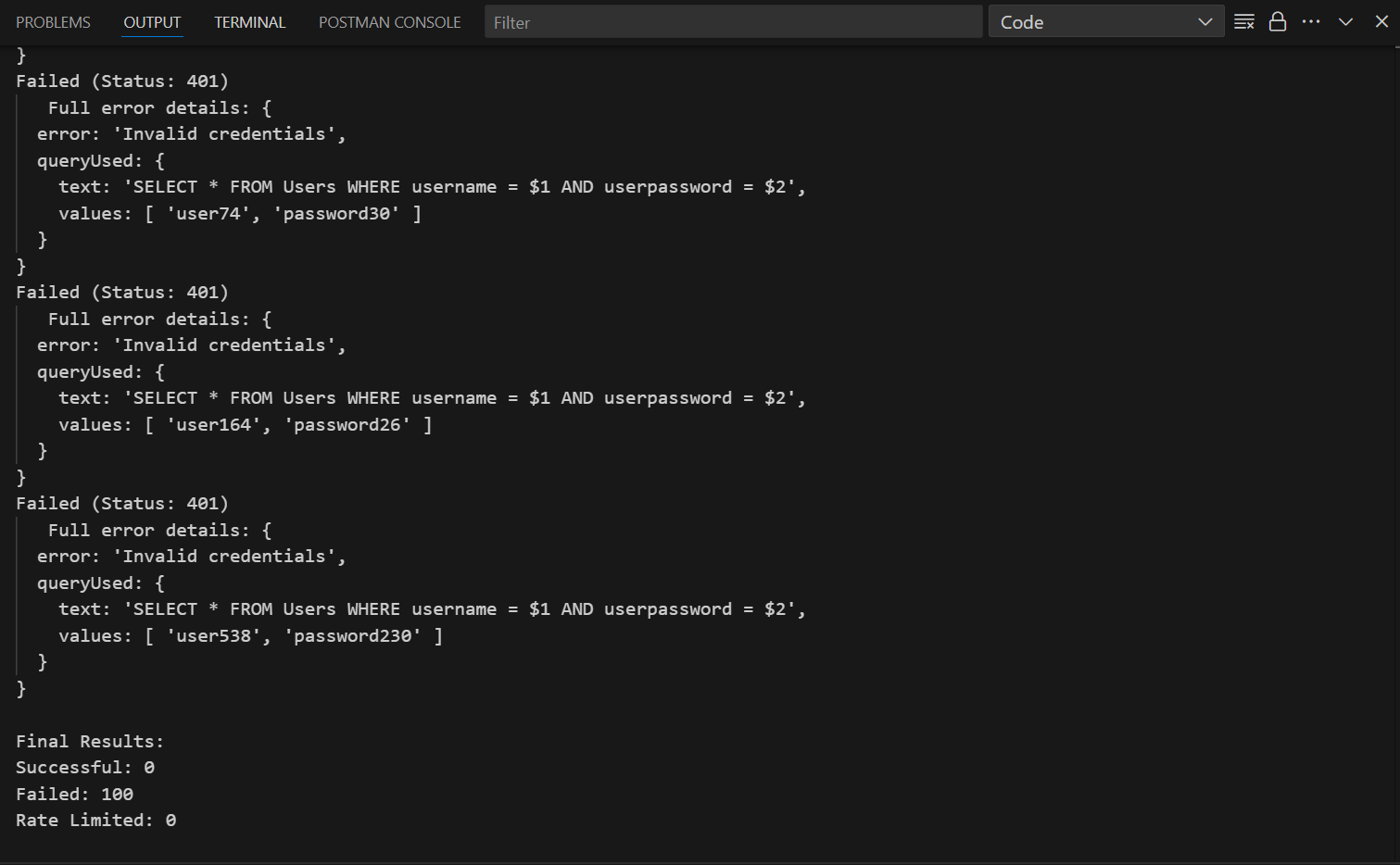
* Sent **100 requests in one minute** from a single IP to overwhelm the server.
* Observed server slowdown and potential crash.

**Solution Implemented:**

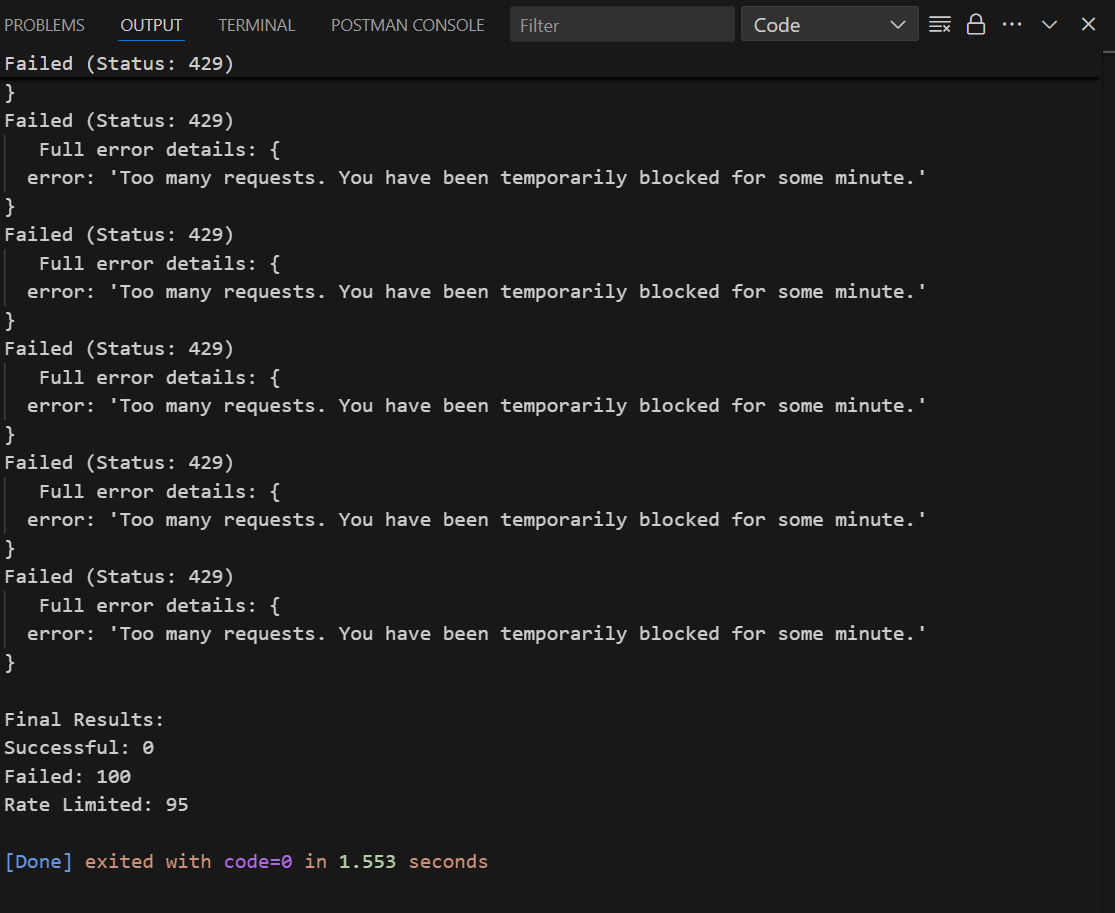
* **Rate Limiting:** Blocked IP if it exceeds **5 requests per minute**.
* **Temporary Blocking:** Banned the IP for **1 minute** upon exceeding the limit.

**Wireframes:**

* **Before Fix:** Server logs showing 100 requests from a single IP.



* **After Fix:** Logs showing blocked IP after 5 requests.



### ****2.1.2 Cross-Site Scripting (XSS) Attack****

**Attack Description:**

* Injected a malicious script in the **username field**:

<script>alert('Collect Your Reward!'); window.location='http://scam-website.com';</script>

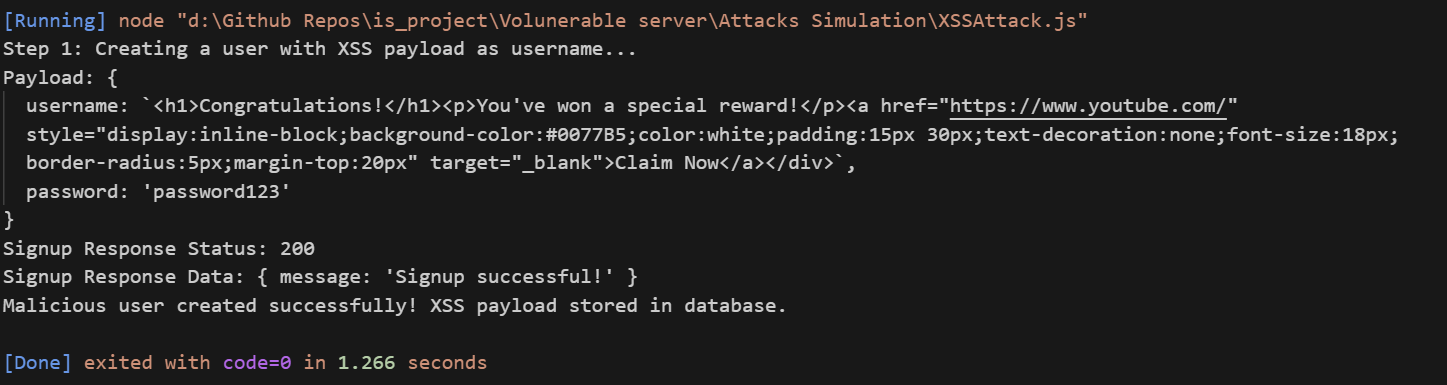
* When rendered, it displayed a fake reward button, redirecting users to a phishing site.

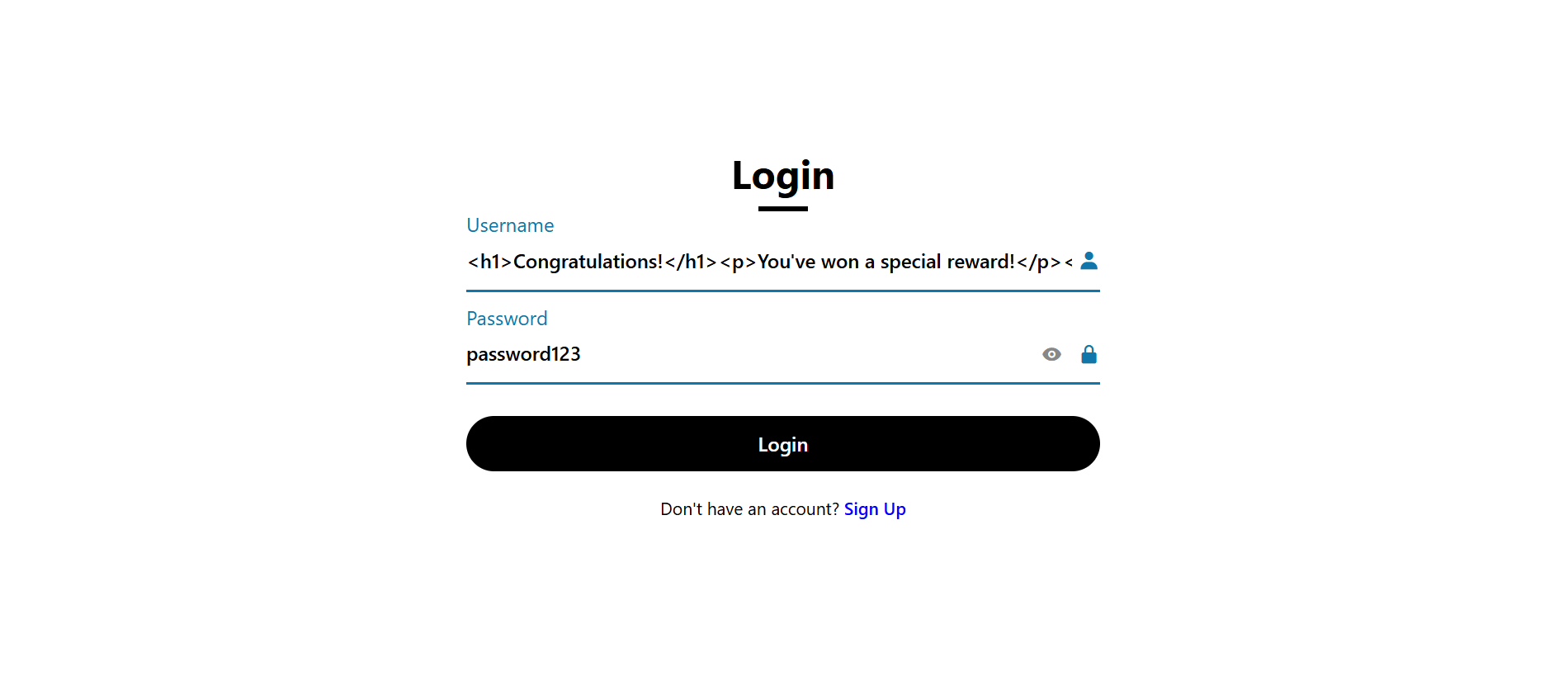
**Solution Implemented:**

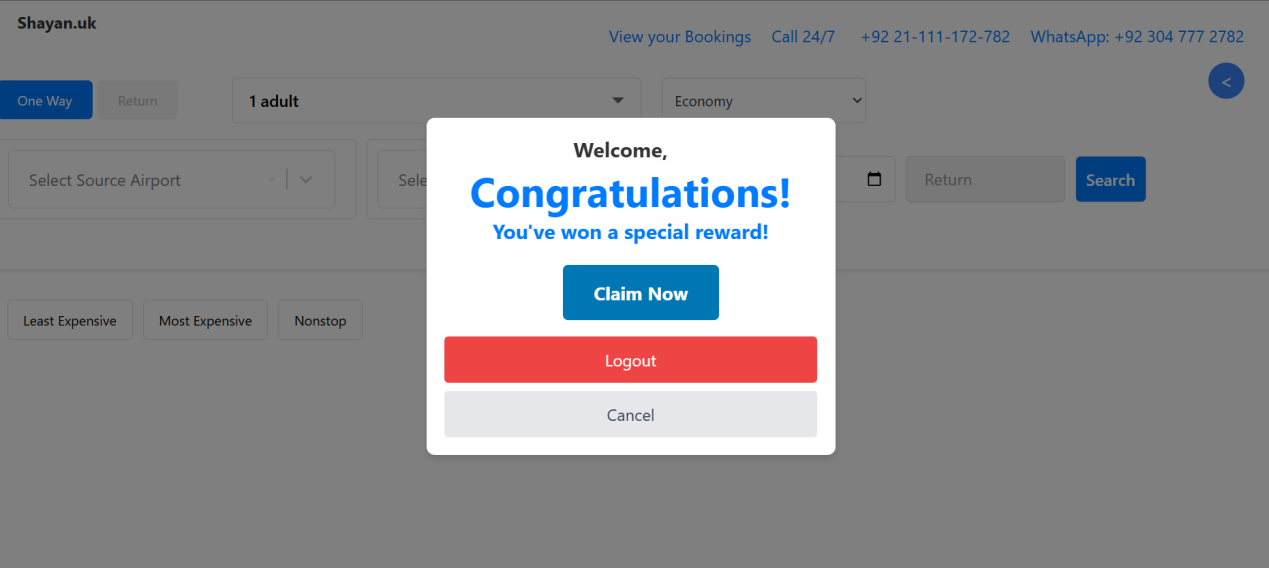
* **Input Sanitization:** Removed HTML/JS tags before database insertion.
* **Output Encoding:** Rendered user inputs as text, not executable scripts.

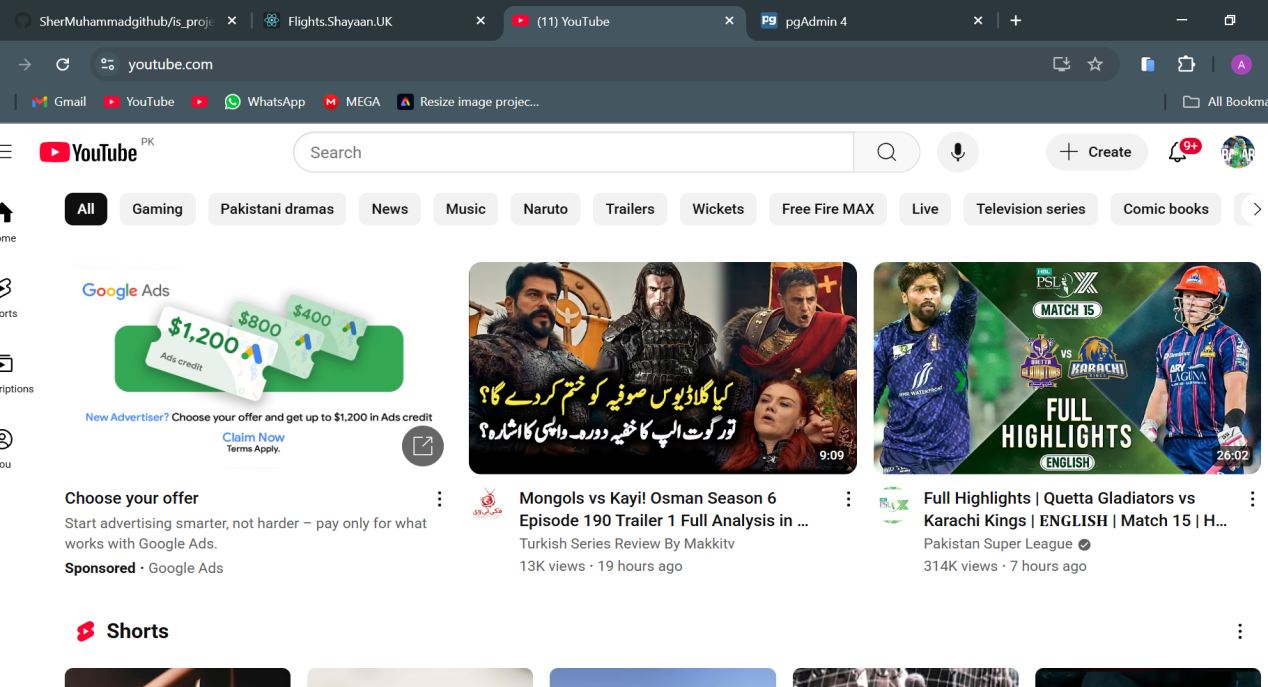
**Wireframes:**

* **Before Fix:** Injected script execution.



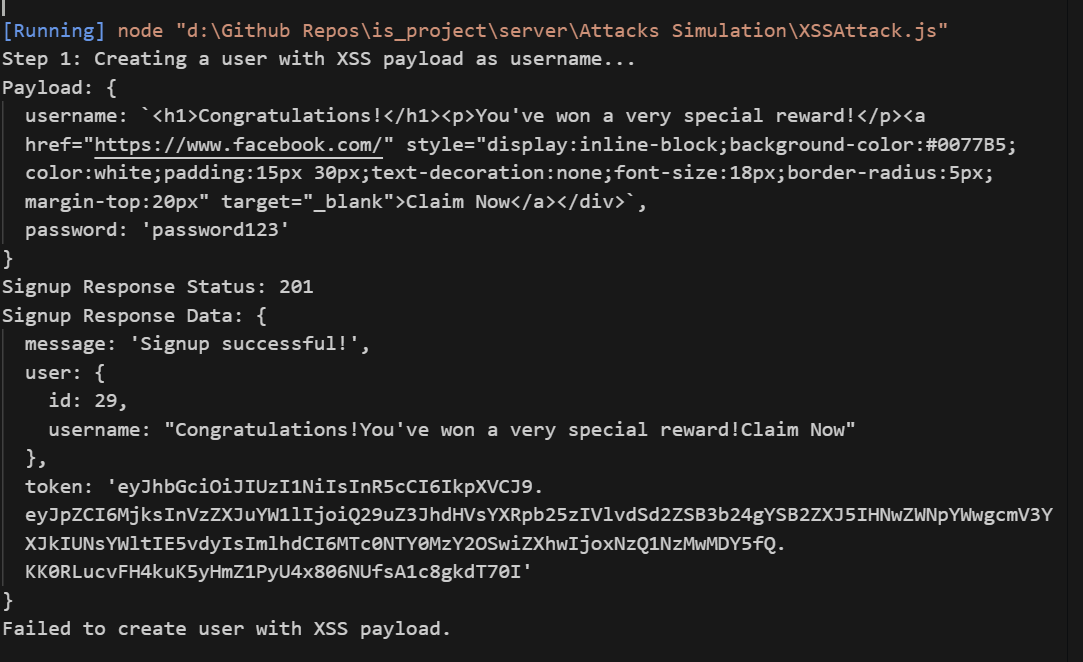






**Unwanted Redirection**

* **After Fix:** Sanitized output (plain text).



### ****2.1.3. Cross-Site Request Forgery (CSRF) Protection****

**Attack Description:**

Without CSRF tokens, attackers could forge requests (e.g., unauthorized fund transfers).

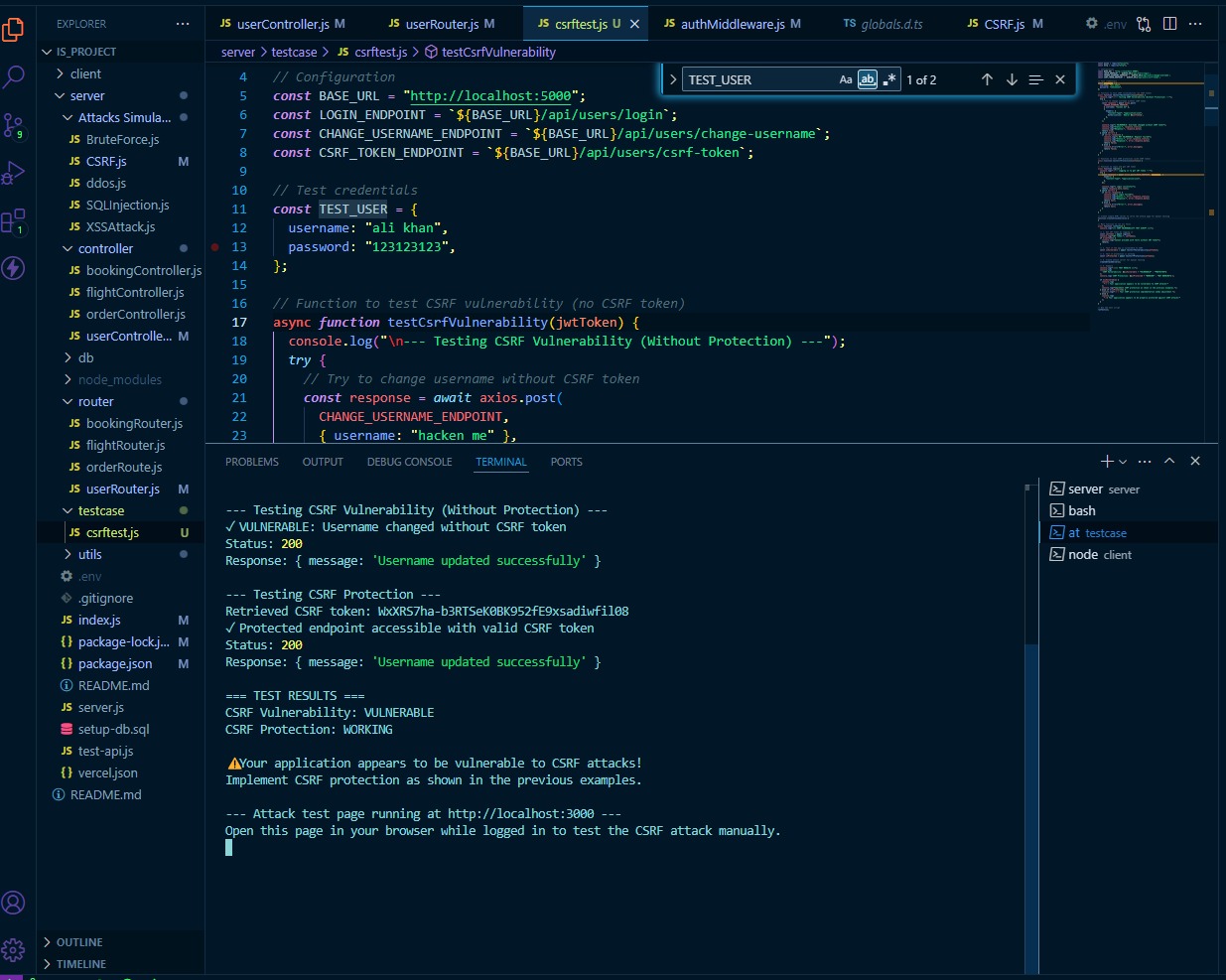
**Solution Implemented:**

**CSRF Tokens:**

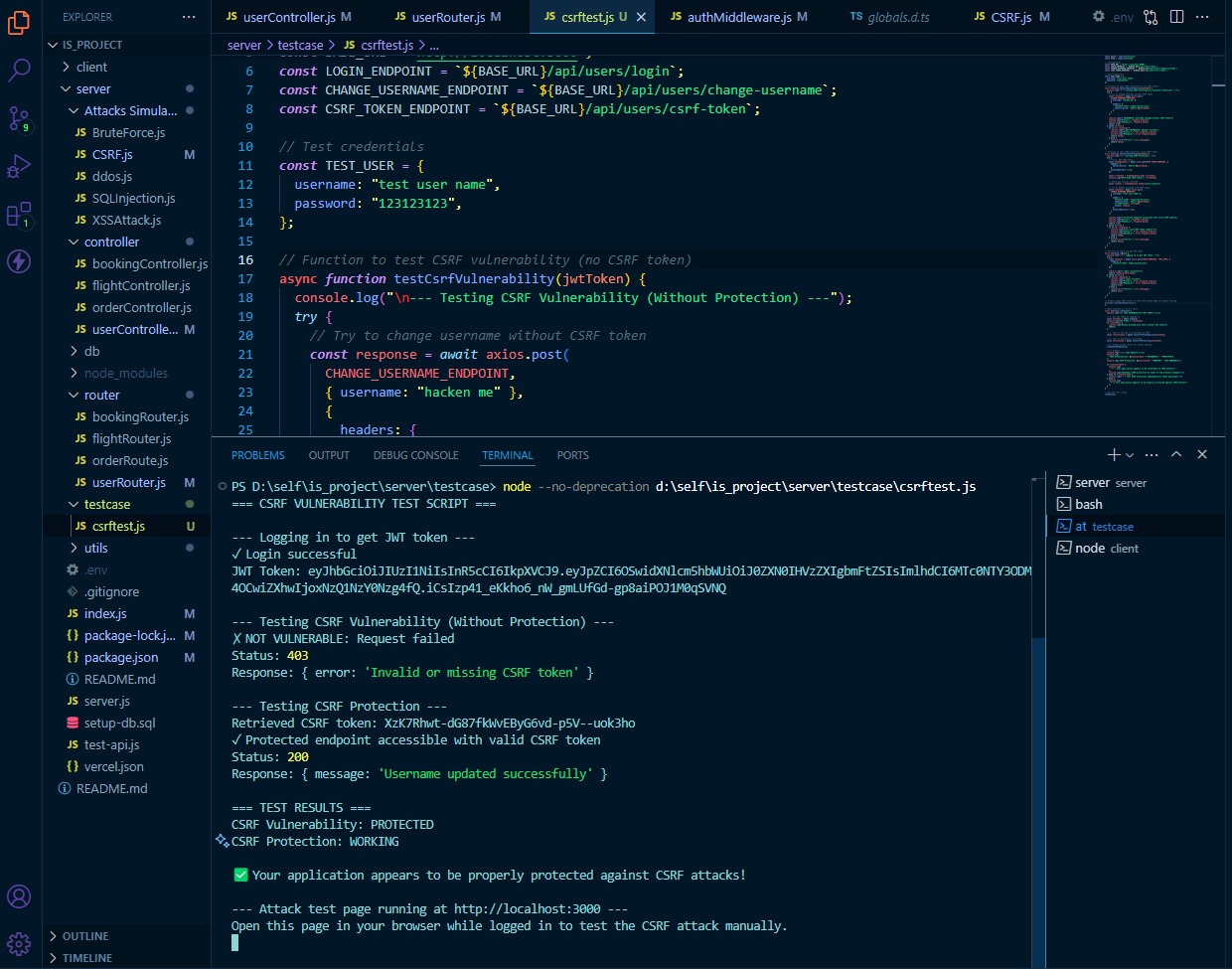
* Generated a unique token upon login.
* Required token for sensitive API requests.
* Rejected requests without valid tokens.

**Wireframes:**

* **Before Fix:** Successful forged request (e.g., POST without token).



* **After Fix:** Request rejected due to missing/invalid token.



### ****2.1.4. Brute Force Password Attack****

**Attack Description:**

* Used a **dictionary of 1000 passwords** to guess a **3-digit PIN**.
* Successfully logged in after multiple attempts.

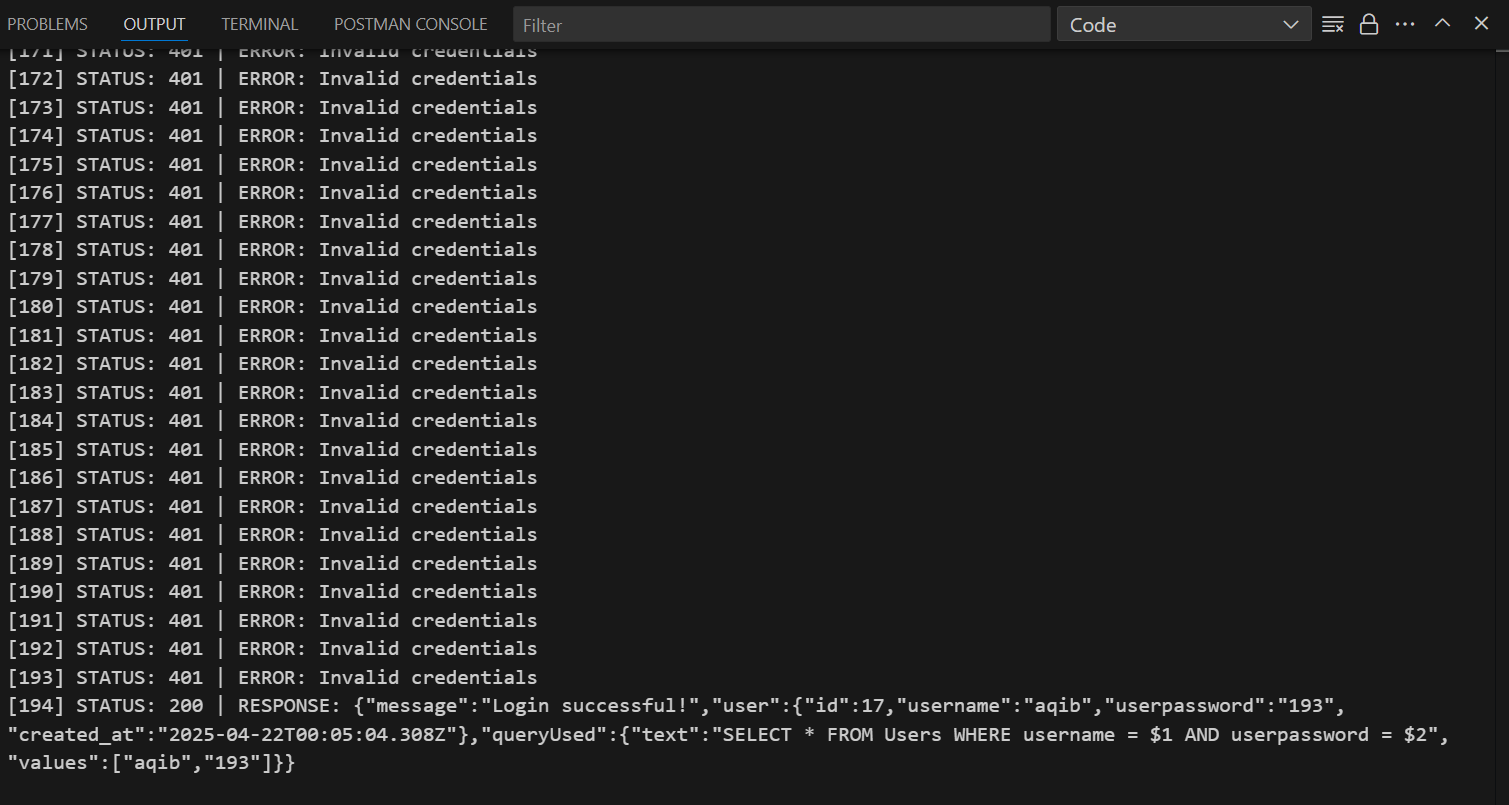
**Solution Implemented:**

**Account Lockout Policy:**

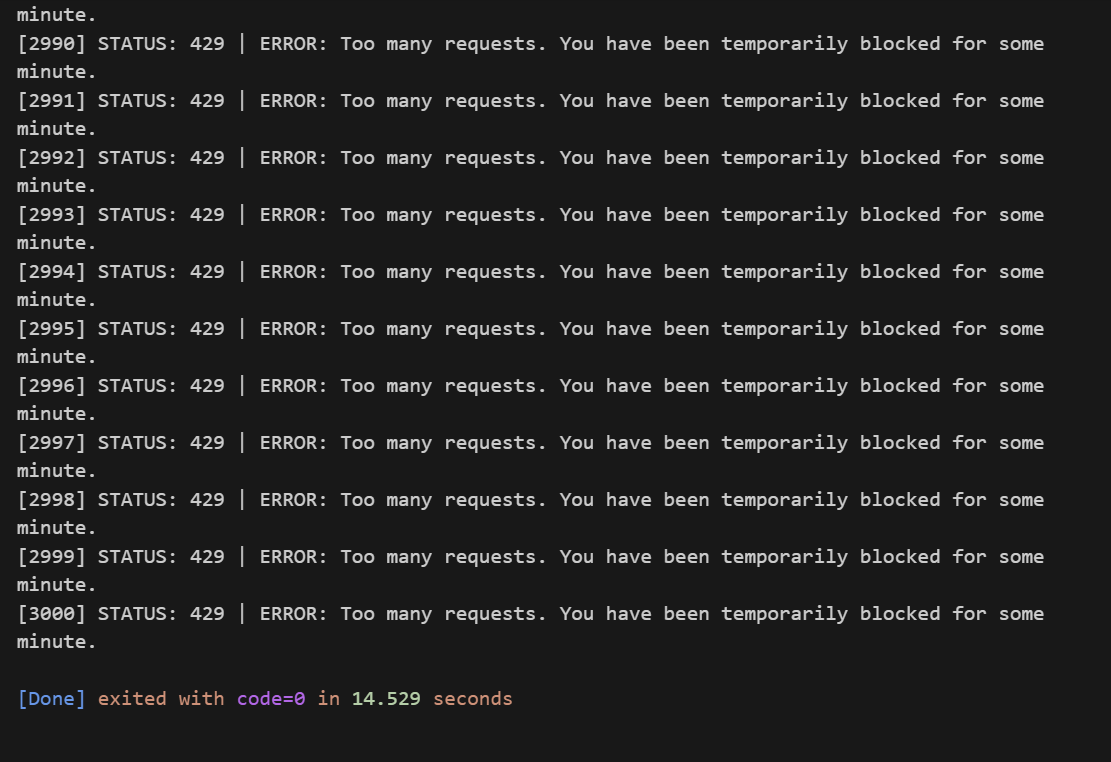
* Blocked user after **5 failed attempts**.
* Lock duration: **5 minutes**.

**Wireframes:**

* **Before Fix:** Successful login after brute-forcing.



* **After Fix:** Account locked after 5 attempts.



### ****2.1.5. SQL Injection Attack****

**Attack Description:**

Injected ' OR 1=1 -- in the login form, bypassing authentication.

**Solution Implemented:**

**Parameterized Queries:**

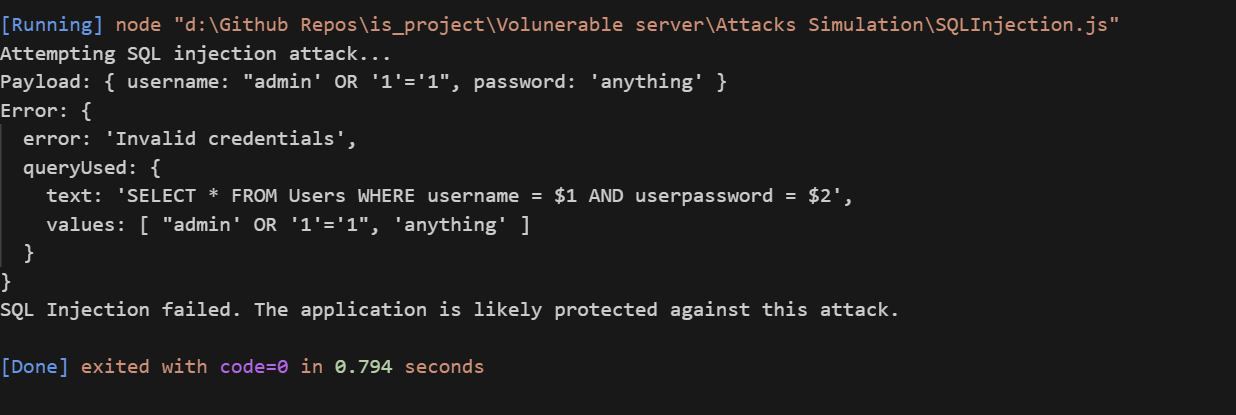
* Used prepared statements to separate SQL logic from user input.

**Wireframes:**

* **Before Fix:** Successful login with SQL injection.



* **After Fix:** Login fails with malicious input.



### ****2.1.6. Location-Based Access Restriction****

**Attack Description:**

Requests from **Israel** were allowed by default.

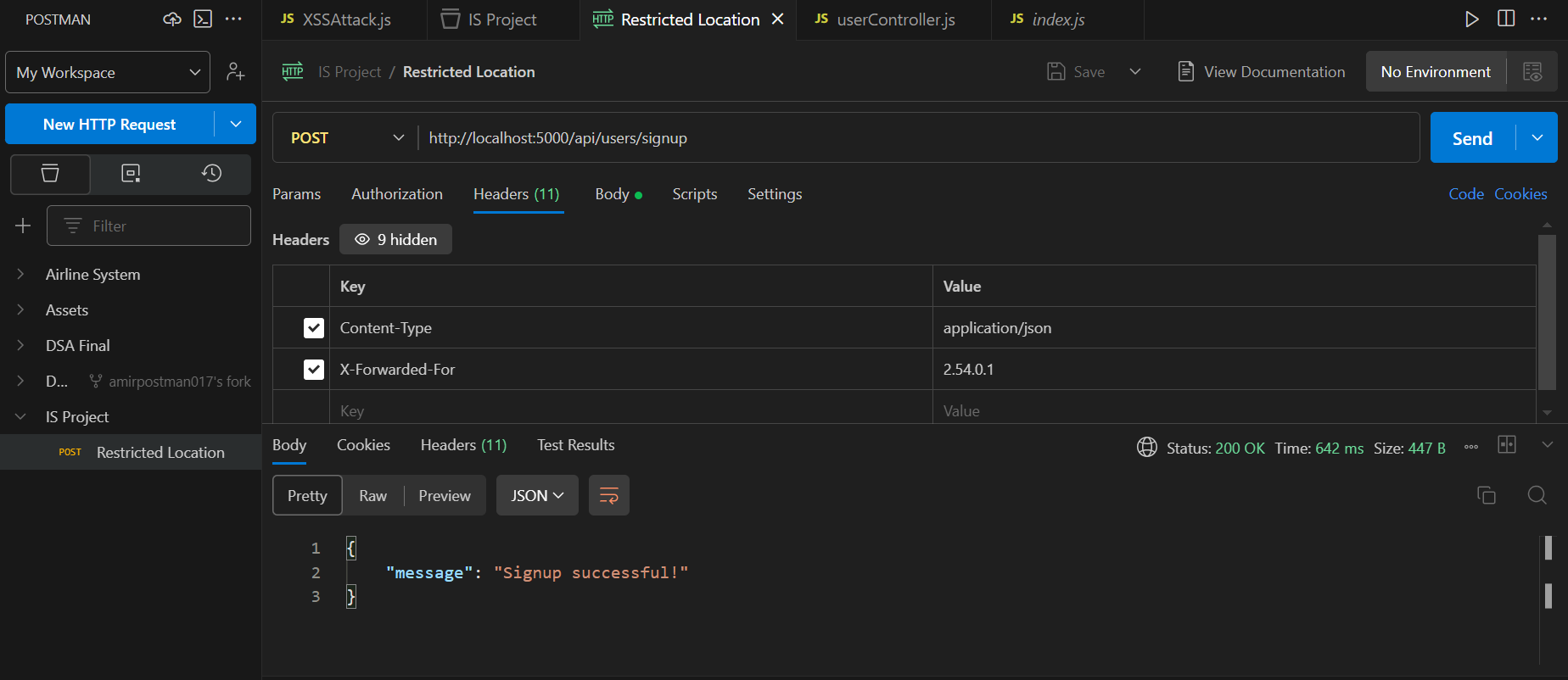
**Solution Implemented:**

**Geo-Blocking:**

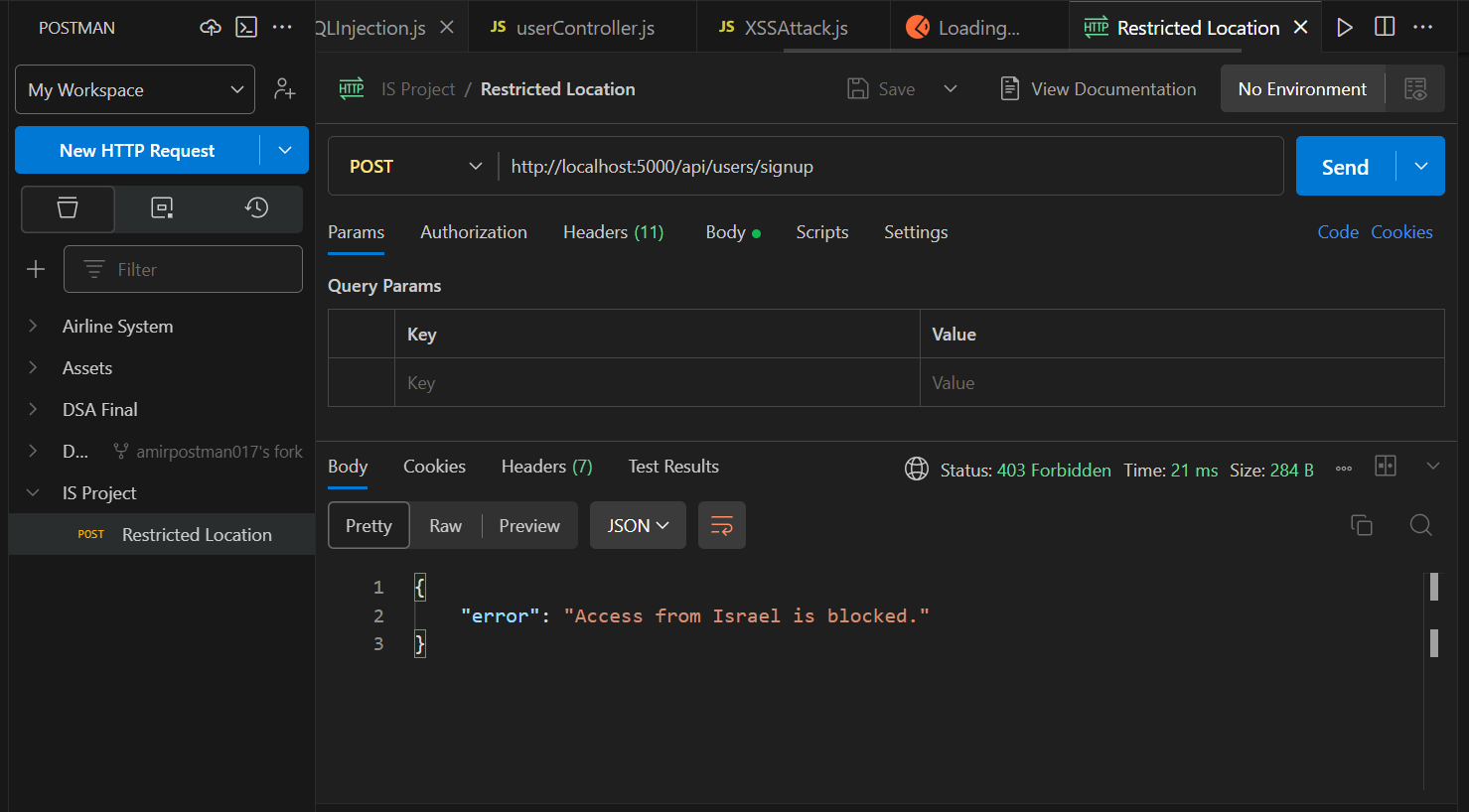
* Detected IP geolocation.
* Blocked requests from **Israel**.

**Wireframes:**

* **Before Fix:** Successful access from Israel.



* **After Fix:** "Access Denied" for Israeli IPs.



### ****2.1.7. Password Encryption (Feistel Cipher)****

**Attack Description:**

Passwords stored in **plaintext**, vulnerable to database leaks.

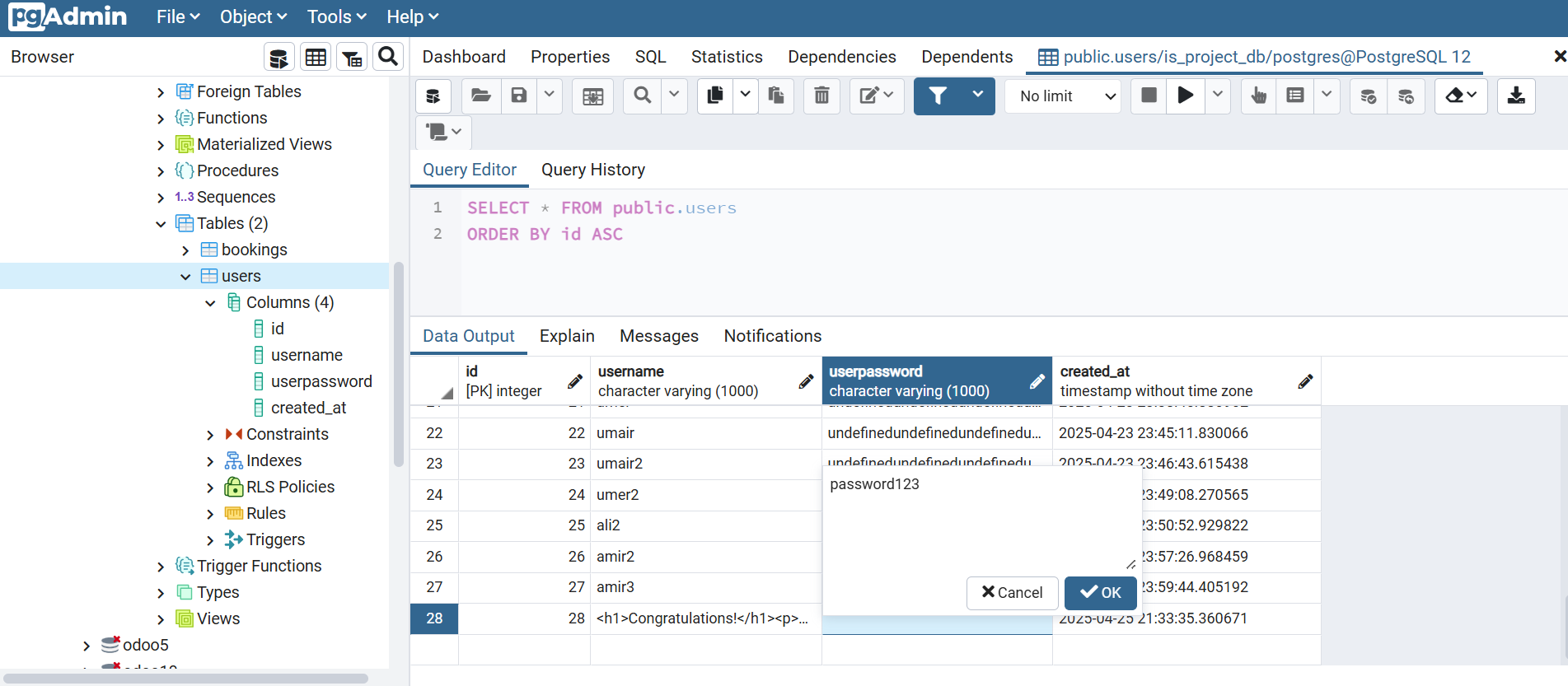
**Solution Implemented:**

**Feistel Cipher Encryption:**

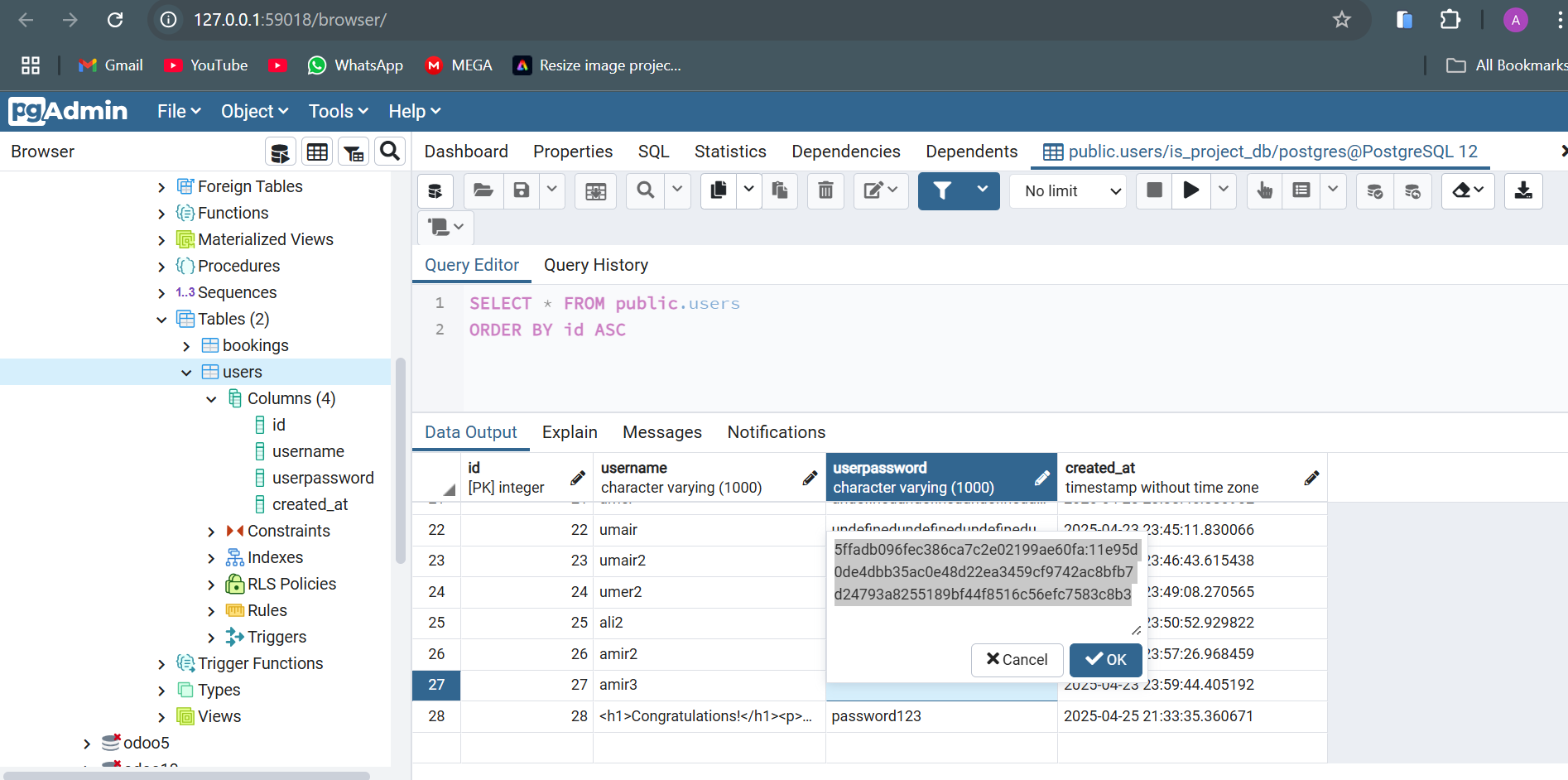
* Encrypted passwords before storage.
* Decrypted only during verification.

**Wireframes:**

* **Before Fix:** Database showing plaintext passwords.



* **After Fix:** Database showing encrypted passwords.



## ****3. Conclusion:****

* **Security is a continuous process** – new threats emerge constantly.
* **Input validation and encryption** are critical.
* **Rate limiting and geo-blocking** can prevent abuse.

## ****4. References:****

**GitHub: <https://github.com/SherMuhammadgithub/is_project>**