VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CRYPT -A- THON

TITLE:-

IOT DEVICE SECURITY FRAMEWORK: CREATE A SECURITY FRAMEWORK FOR IOT DEVICES TO PROTECT AGAINST VULNERABILITIES SUCH AS

UNAUTHORIZED ACCESS, DATA BREACHES, AND DENIAL-OF-SERVICE (DOS) ATTACK

TEAM MEMBERS:

HARIHARANN J

AGASTHYA N

YASWANTH V J

RAGHUL N S

PAVAN P

VISHAL G

Iot Device Security Framework

Information and System Security Crypto-thon

Table of contents

01

Iot Device Security Framework

To create a security framework for IoT devices to protect against vulnerabilities such as unauthorized access, data breaches, and denial-of-service (DoS) attacks.

02

Problem Statement

With the rise of Internet of Things (IoT) devices in our homes, workplaces, and public spaces, keeping them secure has become critical. IoT devices often lack strong security measures, making them easy targets for hackers.

Problem Statement

Unauthorized access

Hackers can take control of IoT devices if they don’t have strong protections, which could lead to misuse of devices and access to private data.

Data Breaches

Sensitive information transmitted by IoT devices can be exposed if it’s not properly secured, risking personal data and business information.

Denial-Of-Service (DOS)

Attackers can overload devices with traffic, causing them to stop working or become inaccessible which is dangerous for critical applications

03

Gap Identification

To identify gaps in an IoT Device Security Framework, we need to examine areas where vulnerabilities might still exist despite the implemented framework, as well as any evolving security challenges.

Gap Identification

Insufficient Identity and Access Management (IAM)

- Lack of Uniform Device Identity Standards
- Inadequate Granularity in Access Control

Limitations in Data Protection Mechanisms

- Inconsistent Encryption Practices
- Lack of Secure Storage for Sensitive Data

Gap Identification

Firmware and Software Security Challenges

- Insufficient OTA Update Validation
- Delayed Patching Processes

Network Security and Traffic Segmentation

- Insufficient Network Segmentation
- Vulnerability to Protocol-Based Attacks

Gap Identification

Weaknesses in Threat Detection - Incident Response

- Inadequate Anomaly Detection for IoT-Specific Attacks
- Lack of Automated Incident Response

Physical Security Vulnerabilities

- Inconsistent Physical Security Measures
- Unsecured Local Interfaces

Gap Identification

Compliance and Regulatory Challenges

- Lack of Industry-Wide Standards
- Insufficient Audit and Verification Processes

Device Lifecycle Management Shortcomings

- Incomplete Decommissioning Processes
- Weaknesses in Secure Device Onboarding

Gap Identification

Emerging Threats and Evolving IoT Attack Vectors

- Vulnerability to Emerging Attacks
- Side-Channel Attacks

04

Objectives for the framework

To develop a comprehensive security framework for Internet of Things (IoT) devices aimed at safeguarding against vulnerabilities, including unauthorized access, data breaches, and denial-of-service (DoS) attacks

Objectives of the framework

01 Access Control and Authentication

02 Data Protection and Privacy

03 Threat Detection and Prevention

04 Resilience Against DOS Attacks

Objectives of the framework

05 Secure Device Management and Updates

06 Compliance and Regulatory Standards

07 Incident Response and Recovery

08 User Awareness and Training

Thank You

ISS

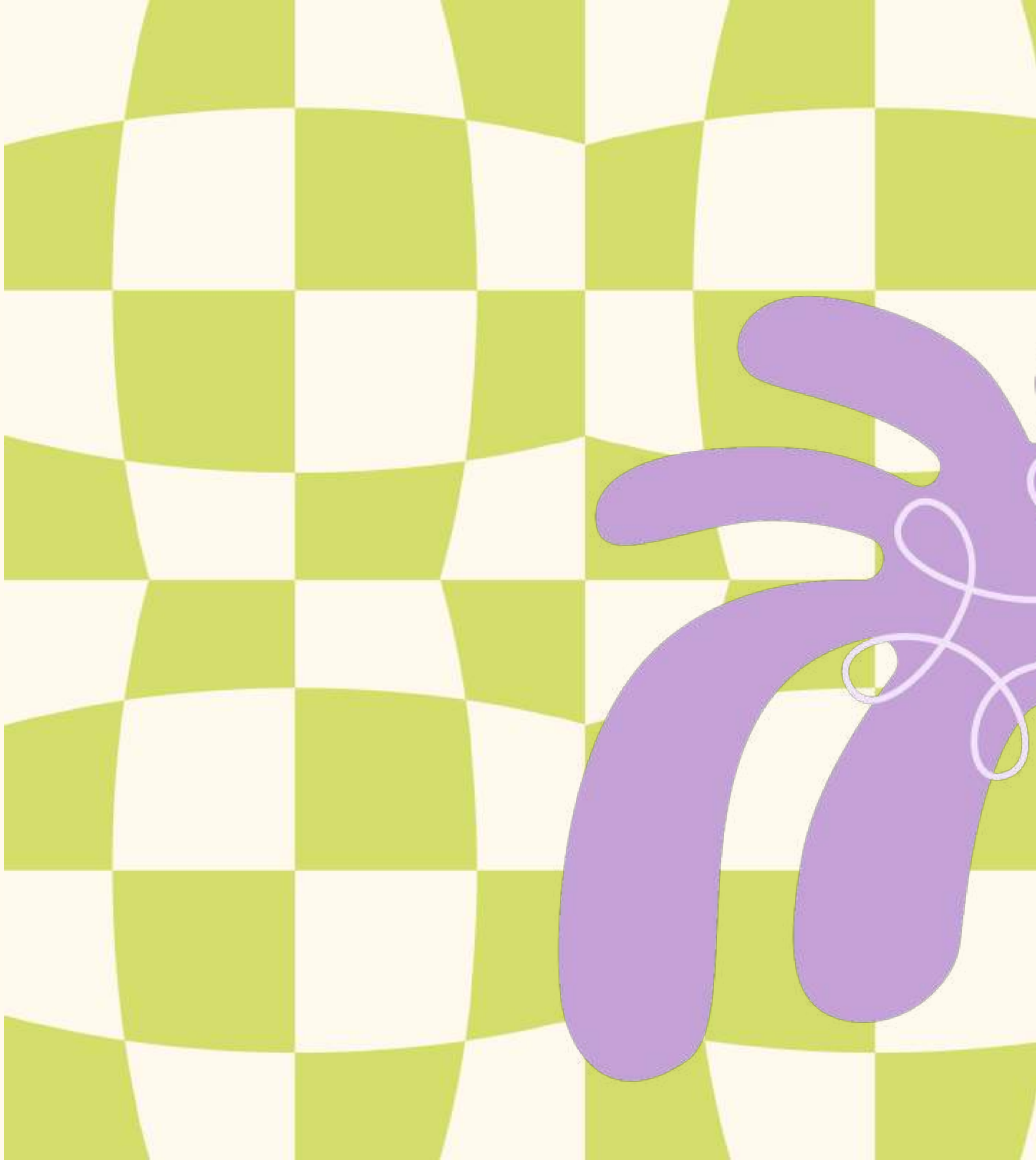
IOT DEVICE SECURITY FRAMEWORK

AGENDA

1 Design methodology

2 Innovation

3 Creativity



DESIGN METHODOLOGY

The steps involved in the methodology is as follows:

Step 1 : Define security requirements

Step 2 : Architectural design

Step 3: Implementation Strategy

Step 4 : Testing and validation

Step 5 : Deployment

Step 6 : Re-evaluation if necessary

DEFINE SECURITY REQUIREMENTS

- Identify Potential threats
- Set security objectives
- Specify security policies

ARCHITECTURAL DESIGN

Authentication Layer

JWT for Token based authentication

Each device has unique credentials

Token issuance for future requests

Data encryption Layer

AES-256 encryption to secure sensitive data at rest and in transit

SHA-256 hashing on critical data to verify data integrity

DoS prevention Layer

Implement rate limiting algorithms like token bucket or leaky bucket

Ip filtering and Blacklisting

Anomaly detection

IMPLEMENTATION

STRATEGY

Authentication Module

JWT for token verification and storing credentials

Cryptographic libraries like PyJWT for token handling.

Encryption and Data Protection Module

AES encryption for data storage

Python libraries for handing encryption and hashing

DoS Mitigation Module

Rate limiter to control request

logging and monitoring system for indication

IP blacklisting

implementation

TESTING AND VALIDATION

UNIT TESTING

PENETRATION TESTING

LOAD AND STRESS TESTING

VALIDATIO N OF SECURITY

DEPLOYMENT

Environment

Visual Studio Code (VSC)

Programming language

Python and Py libraries

Token generation

Postman token generation

RE-EVALUATE SECURITY FRAMEWORK

Improve defense algorithm against unauthorized access, DoS attacks for better optimization

Review and improve the framework if and when necessary

INNOVATION

Zero trust Architecture

Applying a zero-trust model specifically for IoT devices ensures authentication, authorization and validation

Each device communication requires re-authentication and re-verification

Secure Cryptography

Implementing lightweight encryption algorithms tailored for IoT devices.

AES-256 encryption and SHA-256 hashing is implemented

CREATIVITY

Behavior - based access control continuous authentication

Use continuous behavior-based authentication, which tracks typical device usage patterns to recognize and respond to anomalies.

Monitor device behavior, such as data access frequency, operation types, and user interaction patterns

IoT Device Security Framework

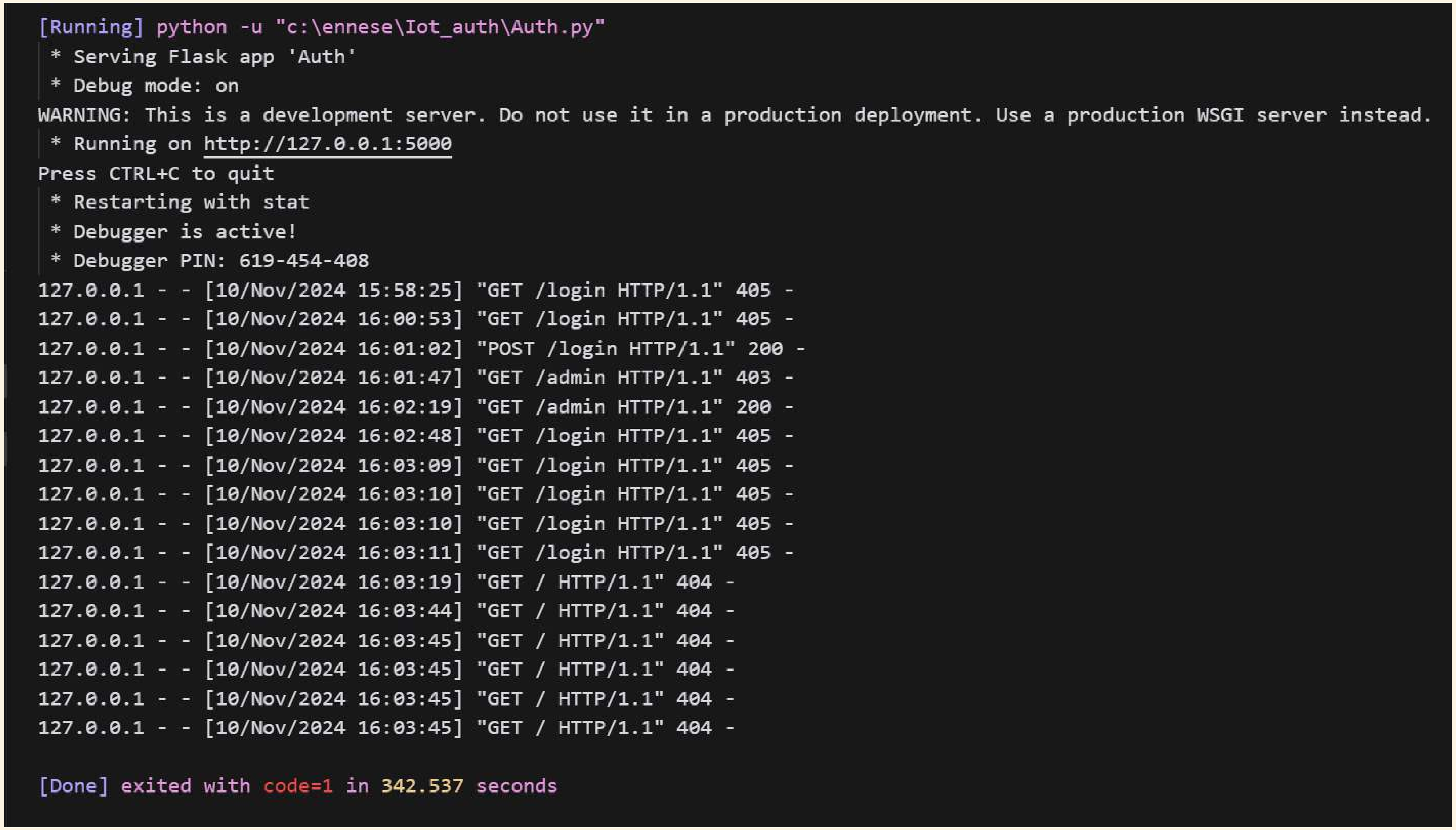
IMPLEMENTATION

Table of contents

01

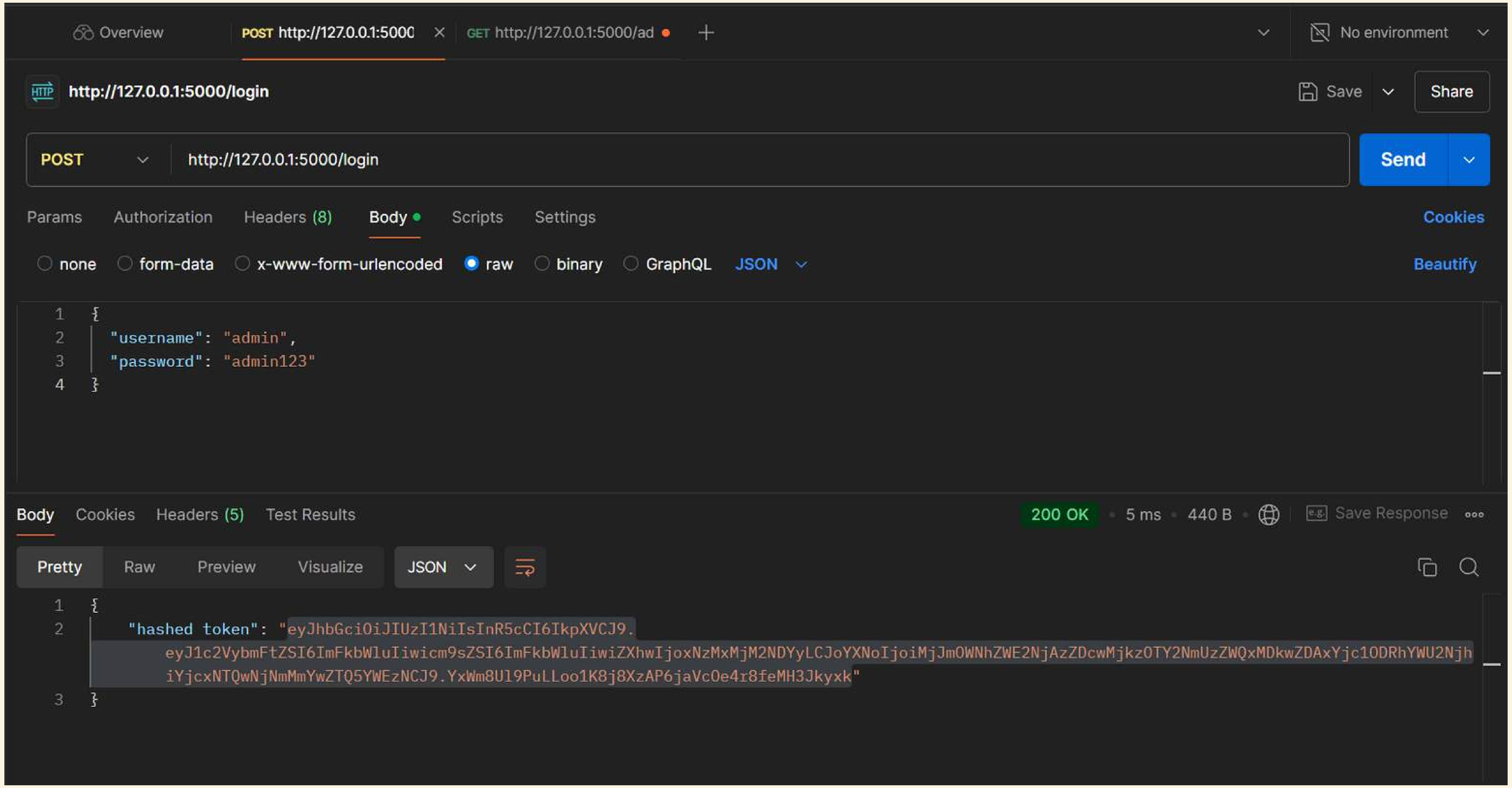
Authentication by JWT

Terminal



```
[Running] python -u "c:\ennese\Iot_auth\Auth.py"
 * Serving Flask app 'Auth'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 619-454-408
127.0.0.1 - - [10/Nov/2024 15:58:25] "GET /login HTTP/1.1" 405 -
127.0.0.1 - - [10/Nov/2024 16:00:53] "GET /login HTTP/1.1" 405 -
127.0.0.1 - - [10/Nov/2024 16:01:02] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [10/Nov/2024 16:01:47] "GET /admin HTTP/1.1" 403 -
127.0.0.1 - - [10/Nov/2024 16:02:19] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [10/Nov/2024 16:02:48] "GET /login HTTP/1.1" 405 -
127.0.0.1 - - [10/Nov/2024 16:03:09] "GET /login HTTP/1.1" 405 -
127.0.0.1 - - [10/Nov/2024 16:03:10] "GET /login HTTP/1.1" 405 -
127.0.0.1 - - [10/Nov/2024 16:03:10] "GET /login HTTP/1.1" 405 -
127.0.0.1 - - [10/Nov/2024 16:03:11] "GET /login HTTP/1.1" 405 -
127.0.0.1 - - [10/Nov/2024 16:03:19] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [10/Nov/2024 16:03:44] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [10/Nov/2024 16:03:45] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [10/Nov/2024 16:03:45] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [10/Nov/2024 16:03:45] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [10/Nov/2024 16:03:45] "GET / HTTP/1.1" 404 -

[Done] exited with code=1 in 342.537 seconds
```

Postman



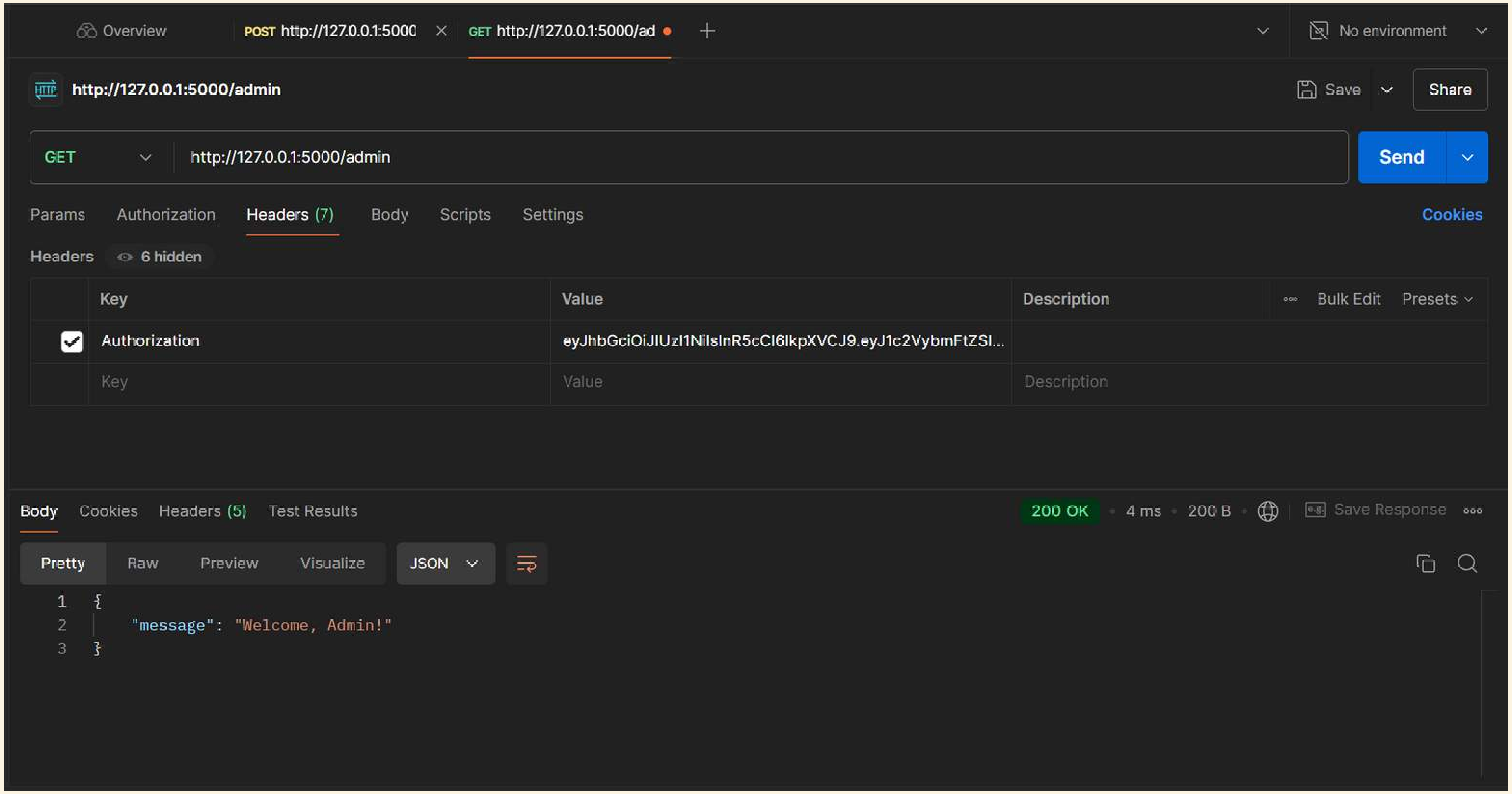
POST http://127.0.0.1:5000/login

```
{
  "username": "admin",
  "password": "admin123"
}
```

200 OK • 5 ms • 440 B

```
{
    "hashed token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ1c2VybmFtZSI6ImFkbWluIiwicm9sZSI6ImFkbWluIiwiZXhwIjoxNzMxMjM2NDYyLCJoYXNoIjoiMjJmOWNhZWE2NjAzZDcwMjkzOTY2NmUzZWQxMDkwZDAxYjc1ODRhYWU2Njh
        iYjcxNTQwNjNmMmYwZTQ5YWEzNCJ9.YxWm8Ul9PuLLoo1K8j8XzAP6jaVcOe4r8feMH3Jkyxk"
}
```

Postman



http://127.0.0.1:5000/admin

GET http://127.0.0.1:5000/admin

Headers

	Key	Value	Description
✓	Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI...	
	Key	Value	Description

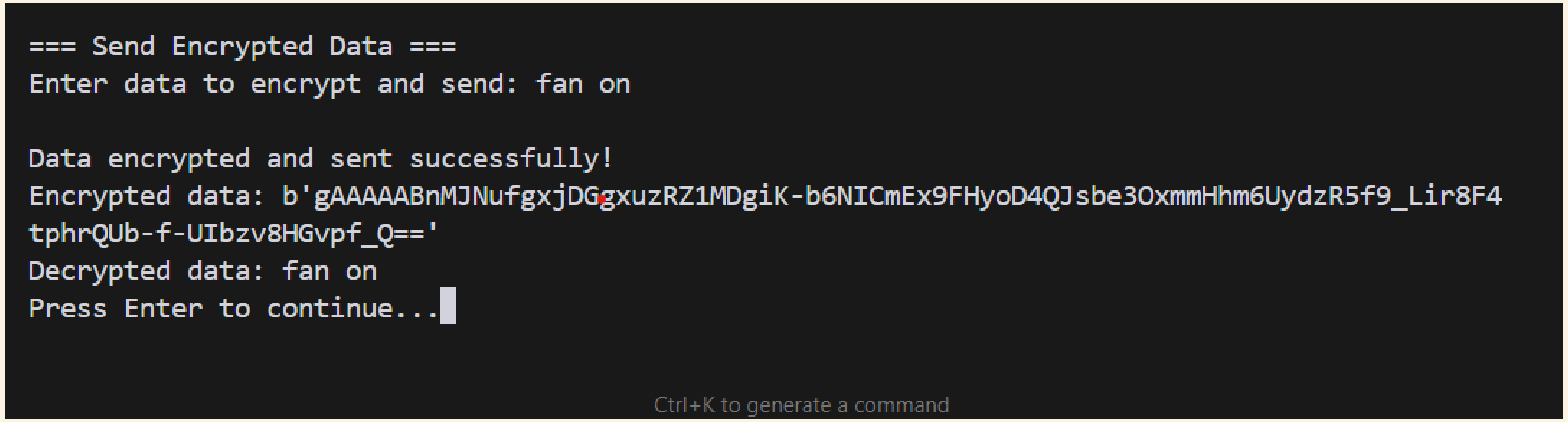
Body — 200 OK · 4 ms · 200 B

```
{
    "message": "Welcome, Admin!"
}
```

02

Data Encryption by AES-256

Terminal

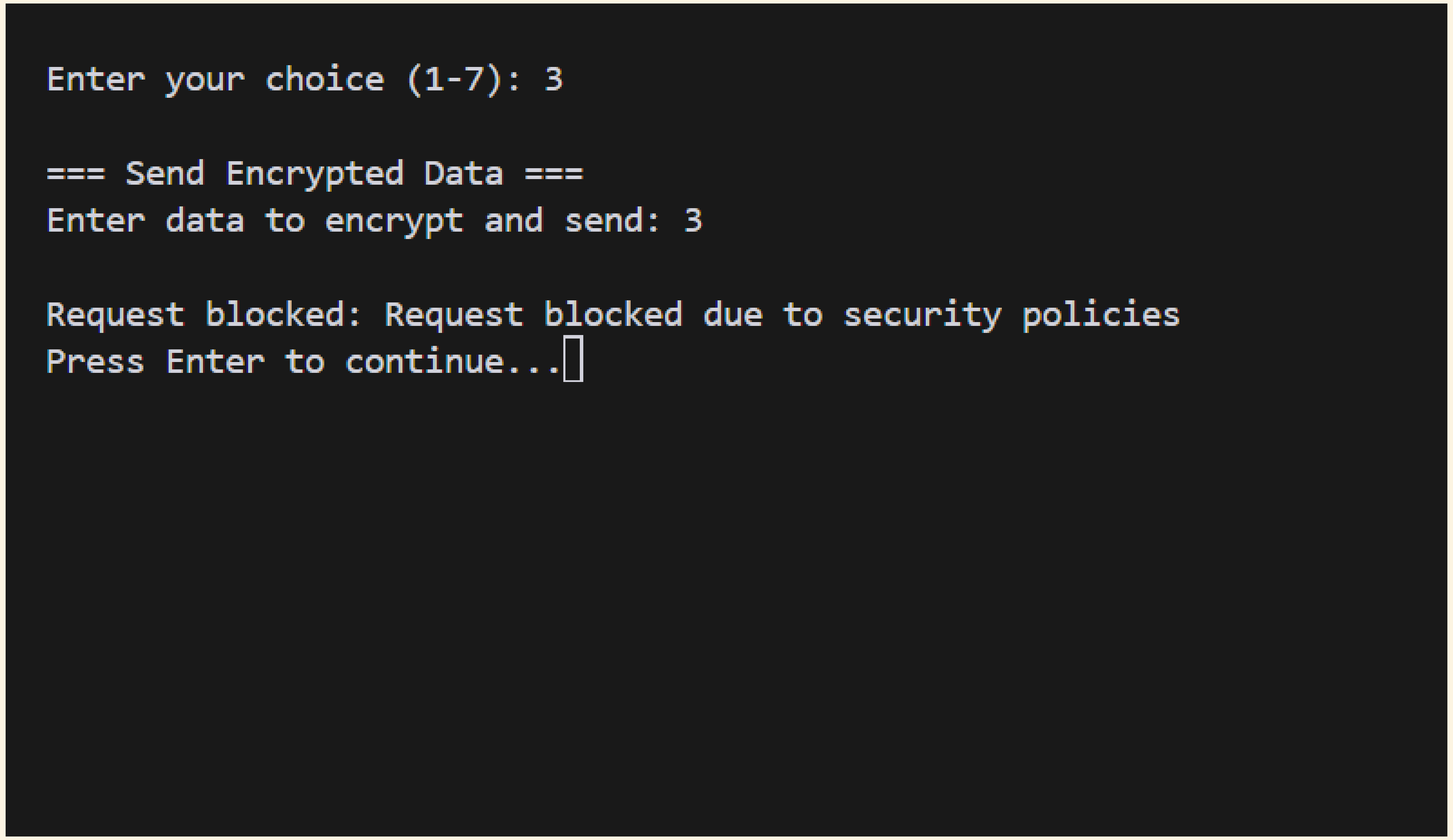


```
=== Send Encrypted Data ===
Enter data to encrypt and send: fan on

Data encrypted and sent successfully!
Encrypted data: b'gAAAAABnMJNufgxjDGgxuzRZ1MDgiK-b6NICmEx9FHyoD4QJsbe3OxmmHhm6UydzR5f9_Lir8F4
tphrQUb-f-UIbzv8HGvpf_Q=='
Decrypted data: fan on
Press Enter to continue...
```

03

DoS prevention

Terminal

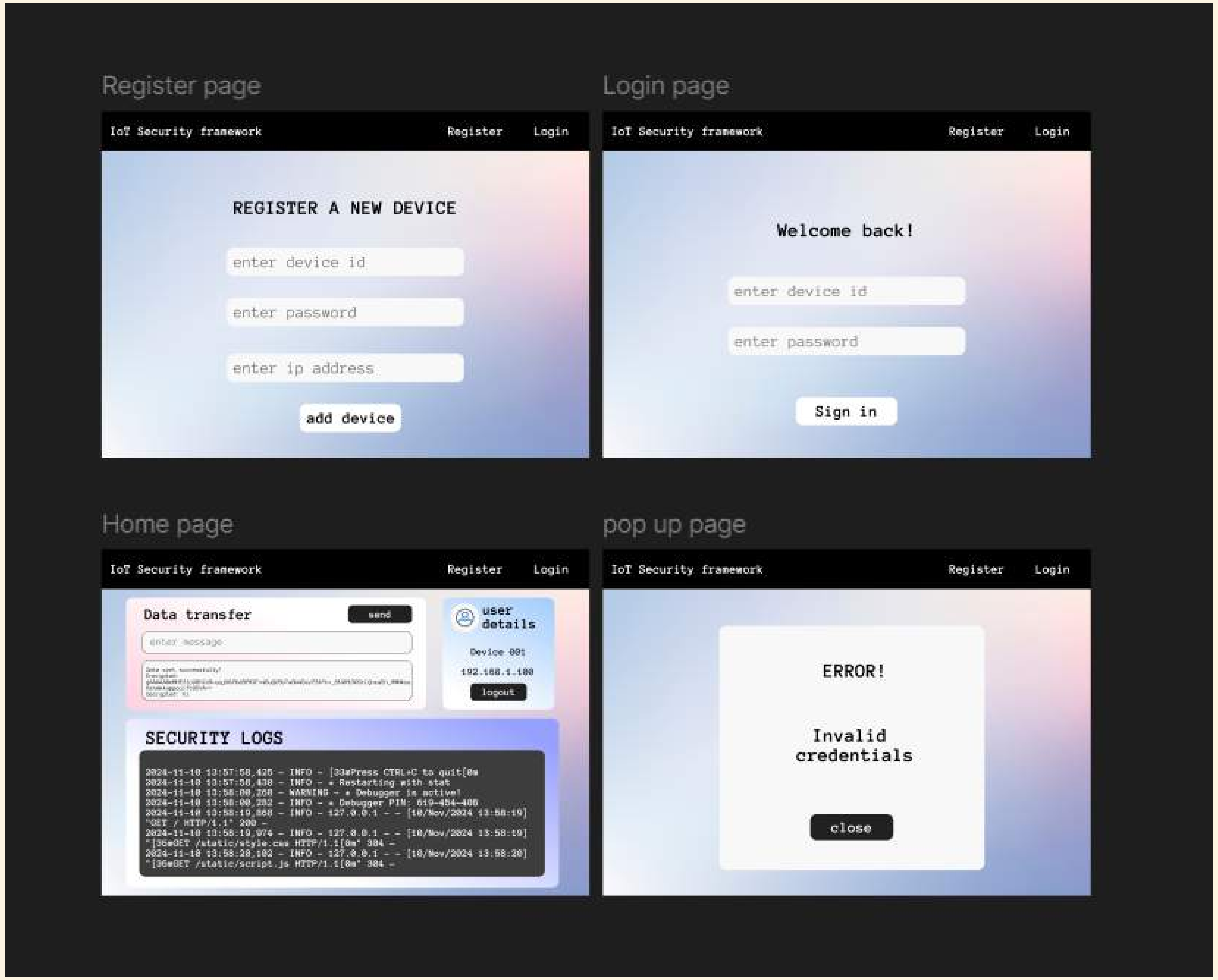


```
Enter your choice (1-7): 3

=== Send Encrypted Data ===
Enter data to encrypt and send: 3

Request blocked: Request blocked due to security policies
Press Enter to continue...
```

04

GUI Designing

Figma



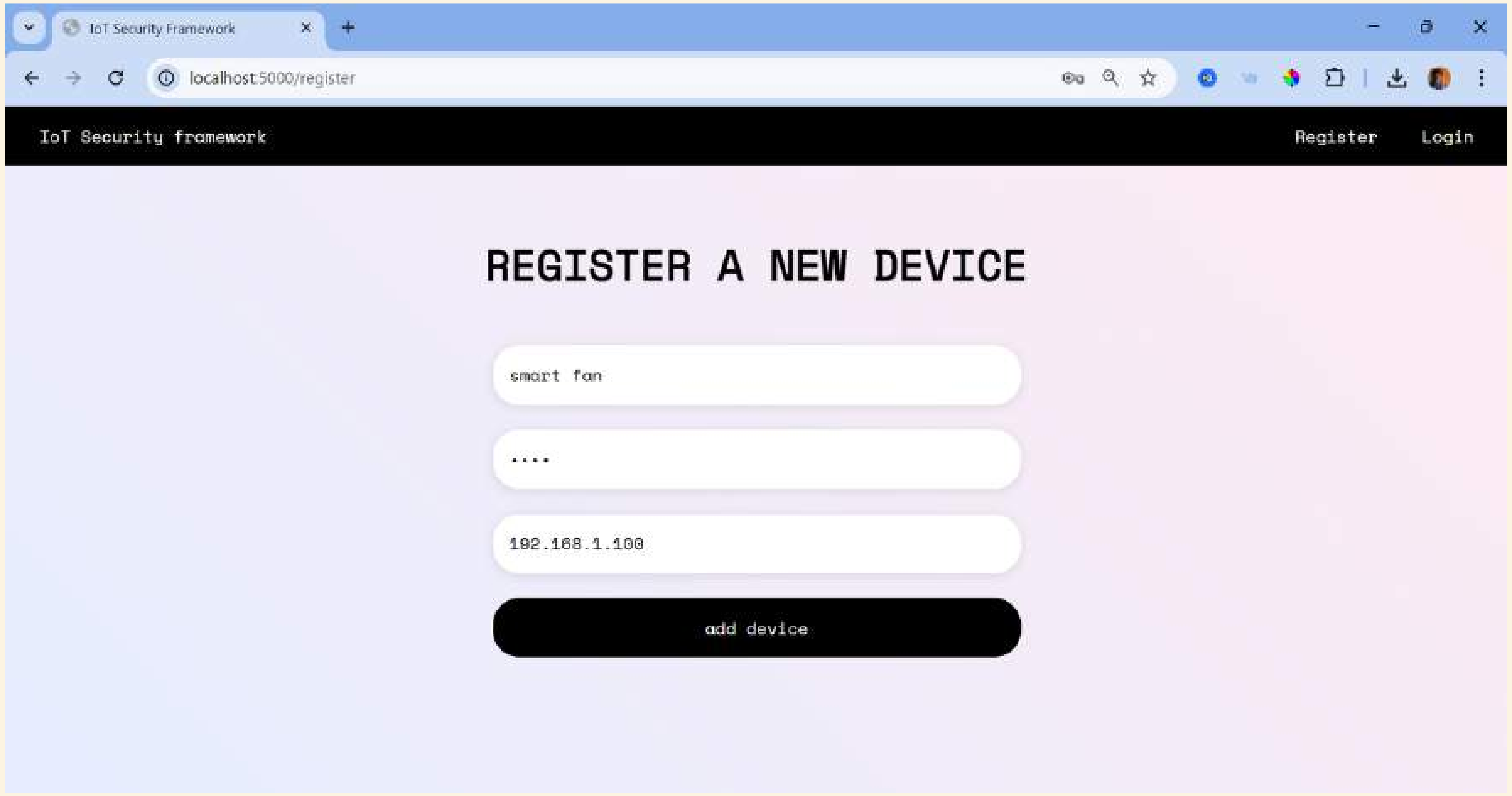
```
Data sent successfully!
Encrypted:
[illegible]
Decrypted: hi
```

```
2024-11-10 13:57:58,425 - INFO - [33mPress CTRL+C to quit[0m
2024-11-10 13:57:58,438 - INFO - * Restarting with stat
2024-11-10 13:58:00,260 - WARNING - * Debugger is active!
2024-11-10 13:58:00,282 - INFO - * Debugger PIN: 619-454-406
2024-11-10 13:58:19,860 - INFO - 127.0.0.1 - - [10/Nov/2024 13:58:19]
"GET / HTTP/1.1" 200 -
2024-11-10 13:58:19,974 - INFO - 127.0.0.1 - - [10/Nov/2024 13:58:19]
"[36mGET /static/style.css HTTP/1.1[0m" 304 -
2024-11-10 13:58:20,102 - INFO - 127.0.0.1 - - [10/Nov/2024 13:58:20]
"[36mGET /static/script.js HTTP/1.1[0m" 304 -
```

05

Interface Hosting(local)

Local Host



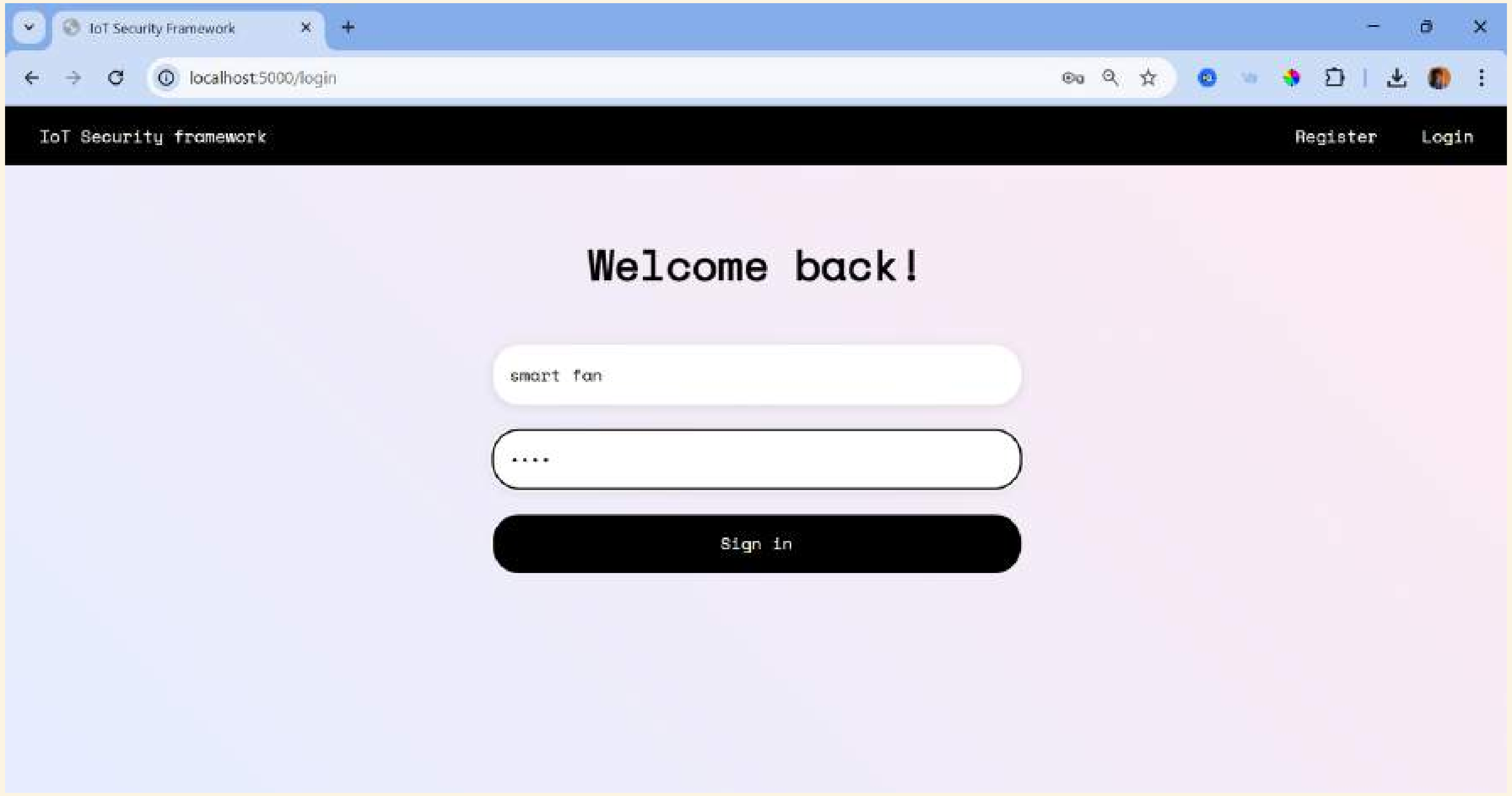
REGISTER A NEW DEVICE

smart fan

····

192.168.1.100

Local Host



IoT Security Framework

localhost:5000/login

IoT Security framework

Register

Login

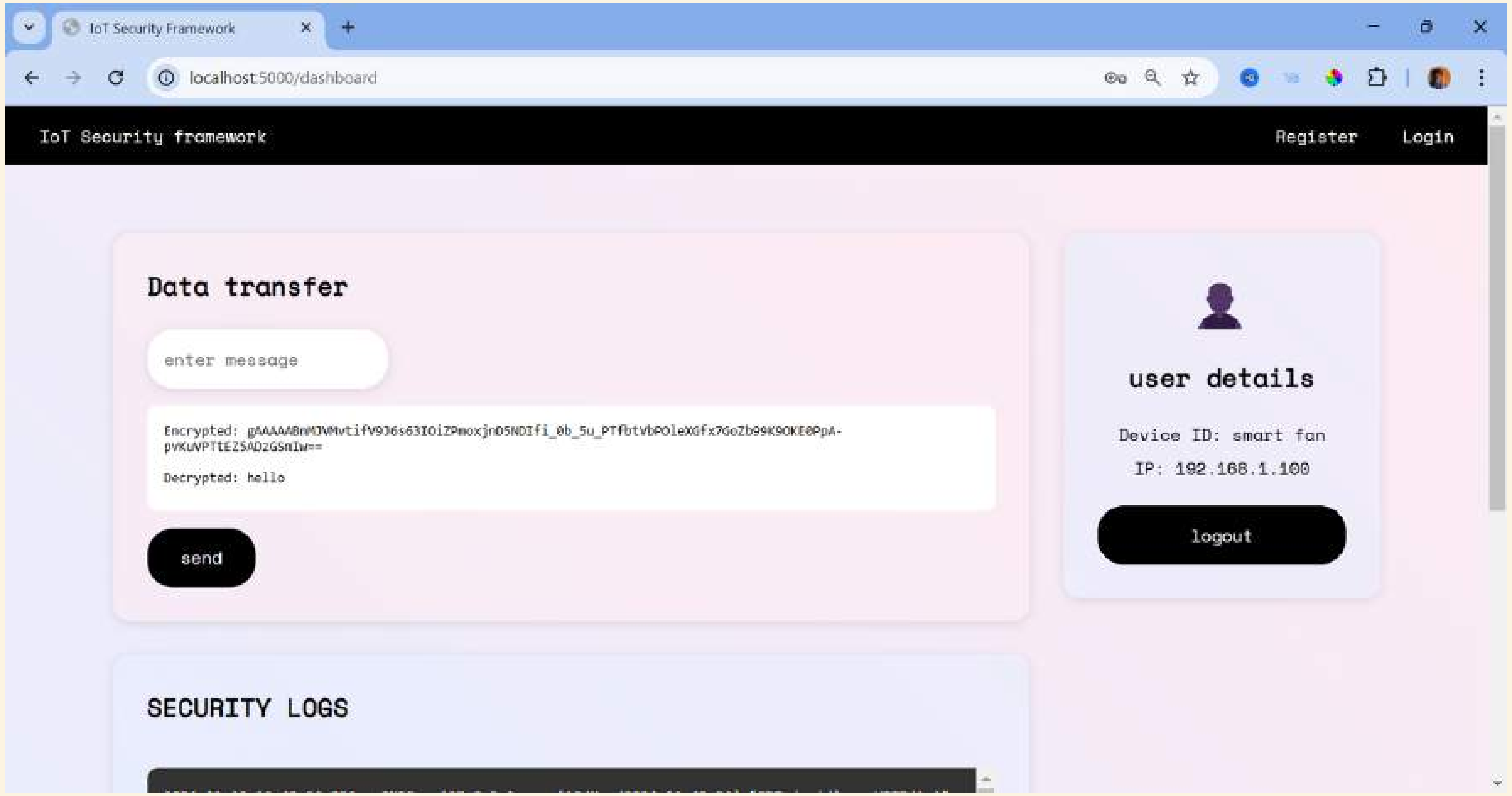
Welcome back!

smart fan

····

Sign in

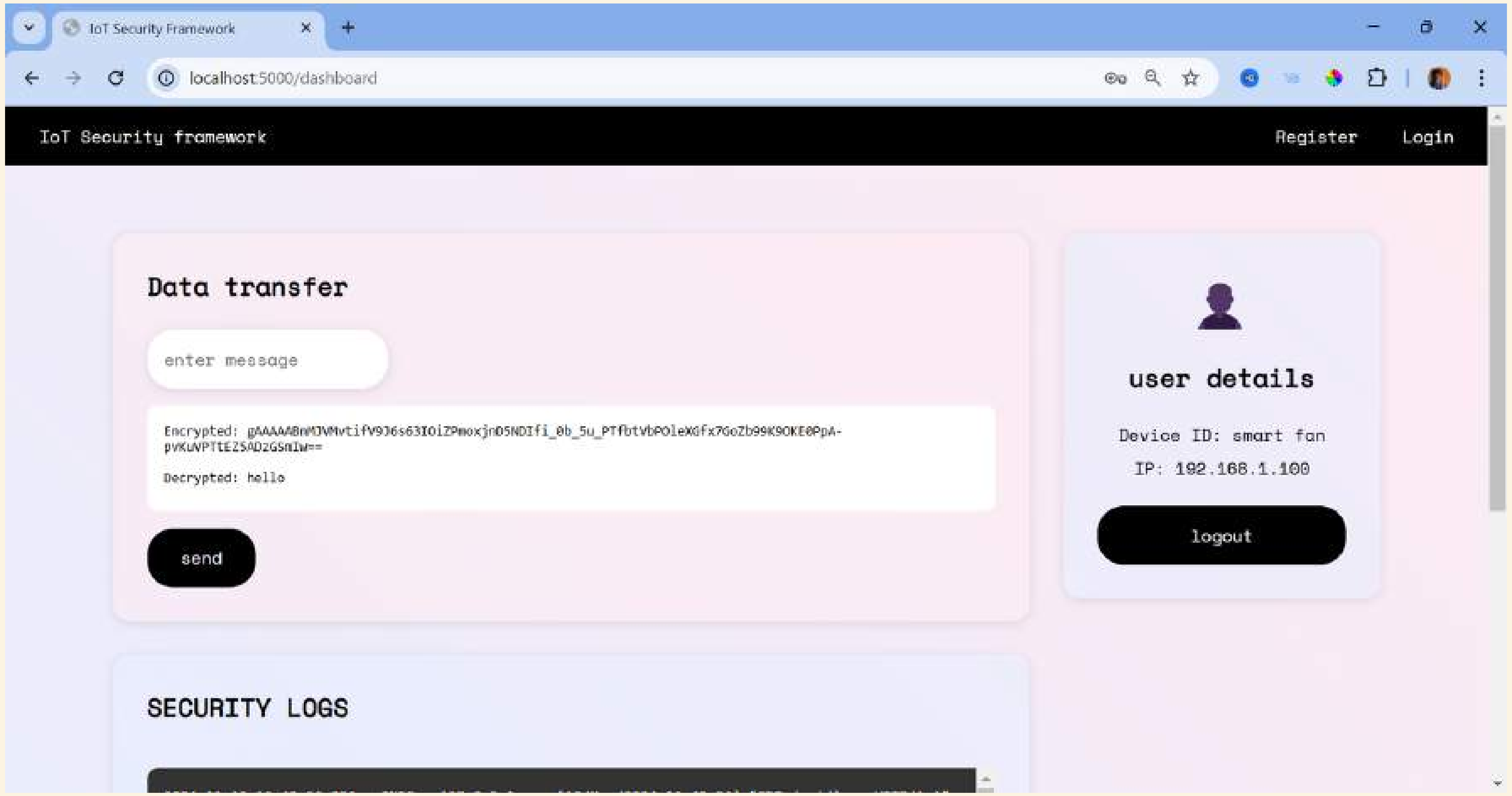
Local Host



```
Encrypted: gAAAAABnMJVMvtifV9J6s63IOiZPmoxjnD5NDIfi_0b_5u_PTfbtVbPOleXGfx7GoZb99K9OKE0PpA-
pVKuVPTtEZ5AD2GSmIw==

Decrypted: hello
```

Device ID: smart fan
IP: 192.168.1.100

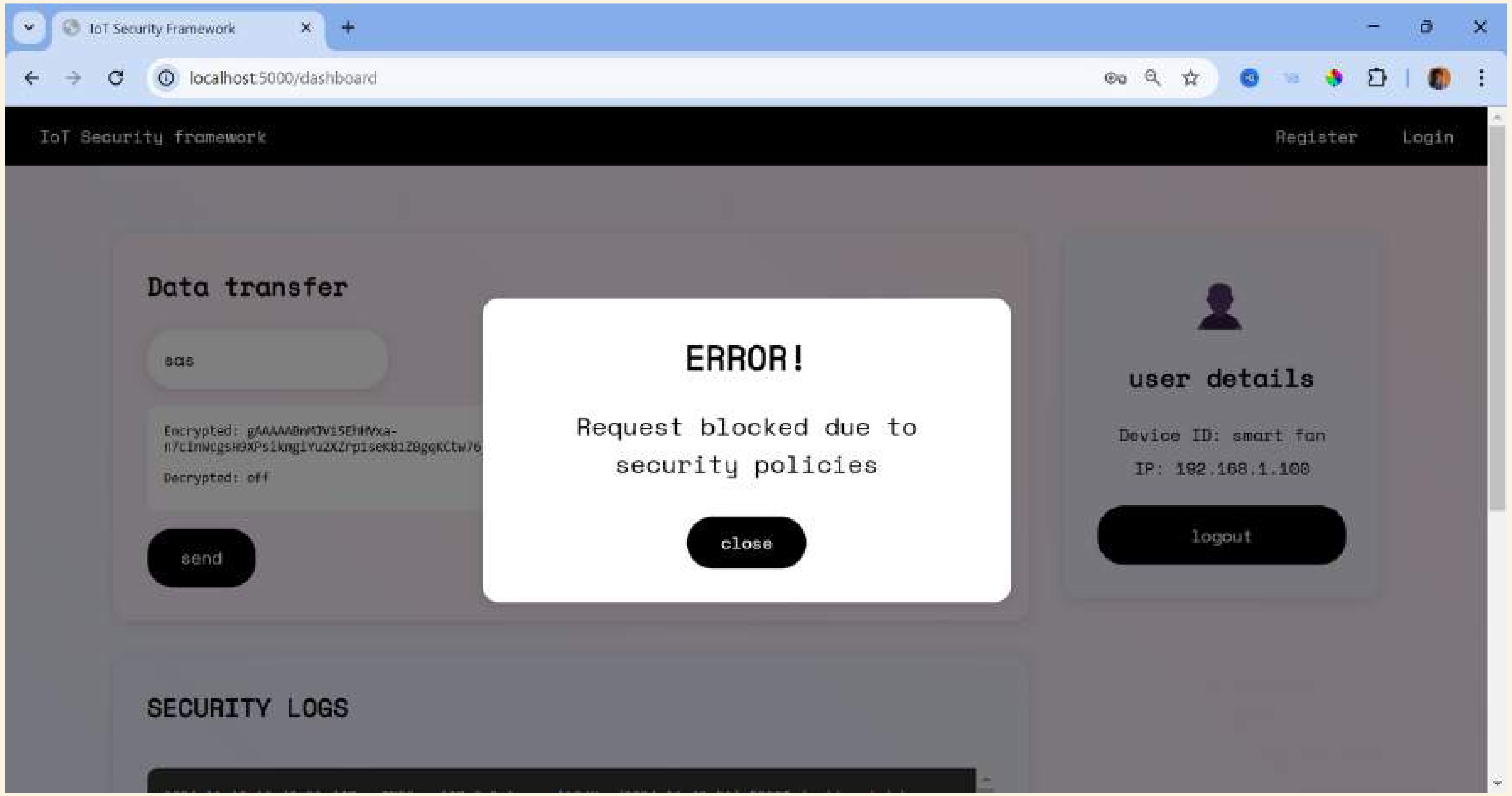
Local Host



```
Encrypted: gAAAAABnMJVMvtifV9J6s63IOiZPmoxjnD5NDIfi_0b_5u_PTfbtVbPOleXGfx7GoZb99K9OKE0PpA-
pVKuVPTtEZ5ADzGSmlw==

Decrypted: hello
```

Device ID: smart fan
IP: 192.168.1.100

Local Host



Data transfer

sas

```
Encrypted: gAAAAABnMOVi5EhHVxa-
n7cInWcgsH9XPslkmgiYu2XZrplseK81Z8gqKCtw76
Decrypted: off
```

ERROR!

Request blocked due to security policies

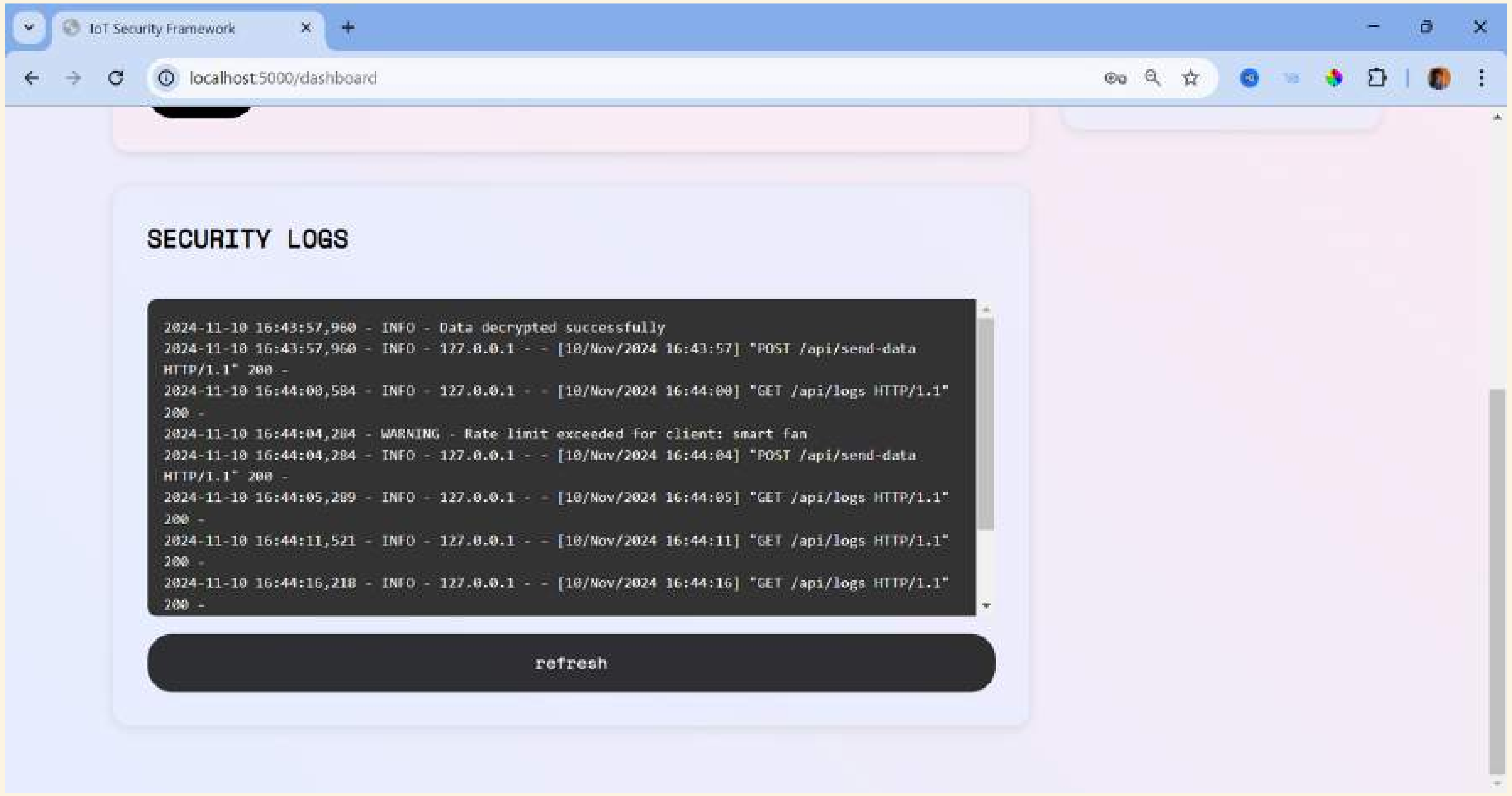
user details

Device ID: smart fan

IP: 192.168.1.100

SECURITY LOGS

Local Host



SECURITY LOGS

```
2024-11-10 16:43:57,960 - INFO - Data decrypted successfully
2024-11-10 16:43:57,960 - INFO - 127.0.0.1 - - [10/Nov/2024 16:43:57] "POST /api/send-data
HTTP/1.1" 200 -
2024-11-10 16:44:00,584 - INFO - 127.0.0.1 - - [10/Nov/2024 16:44:00] "GET /api/logs HTTP/1.1"
200 -
2024-11-10 16:44:04,284 - WARNING - Rate limit exceeded for client: smart fan
2024-11-10 16:44:04,284 - INFO - 127.0.0.1 - - [10/Nov/2024 16:44:04] "POST /api/send-data
HTTP/1.1" 200 -
2024-11-10 16:44:05,289 - INFO - 127.0.0.1 - - [10/Nov/2024 16:44:05] "GET /api/logs HTTP/1.1"
200 -
2024-11-10 16:44:11,521 - INFO - 127.0.0.1 - - [10/Nov/2024 16:44:11] "GET /api/logs HTTP/1.1"
200 -
2024-11-10 16:44:16,218 - INFO - 127.0.0.1 - - [10/Nov/2024 16:44:16] "GET /api/logs HTTP/1.1"
200 -
```

06

Findings and Conclusion

Findings of this Framework

Vulnerabilities

IoT devices are often small, with limited processing power and storage, making them more vulnerable to cyberattacks. Many of these devices lack strong security measures, like encryption or robust authentication.

Diversity of devices

IoT networks often include many types of devices (sensors, cameras, appliances) that communicate differently, making it challenging to secure them all consistently.

Threats

Common threats to IoT networks include malware, unauthorized access, and denial-of-service attacks, which can lead to data leaks, device hijacking, and network downtime.

Conclusion

Layered Security Approach

Implementing a multi-layered security approach is essential, covering everything from individual devices to the network they communicate through. This includes using firewalls, encryption, and regular software updates.

Authentication and Access Control

Strengthening how devices and users access the network, such as requiring passwords and multi-factor authentication, helps prevent unauthorized access.

Monitoring and Maintenance

Continuously monitoring IoT networks and devices allows for the early detection of unusual activity, making it easier to respond to potential threats.

Standardization

Developing and following industry standards for IoT security ensures that all devices and networks have basic protections, helping to close security gaps.

CODE:

Authorization:

```
from flask import Flask, jsonify, request, abort

from functools import wraps

import jwt

import datetime

import hashlib


app = Flask(__name__)

app.config['SECRET_KEY'] = 'your_secret_key'  # Keep this secret

app.config['HASH_SECRET'] = 'another_secret_key'  # Extra key for hashing


# Dummy user data with roles

users = {

    "admin": {"password": "admin123", "role": "admin"},

    "user": {"password": "user123", "role": "user"}

}


# Hash function for additional security

def hash_username(username):

    hash_object = hashlib.sha256((username + app.config['HASH_SECRET']).encode())

    return hash_object.hexdigest()
```

```
# Generate JWT token with hashed username
def generate_token(username, role):
    hashed_username = hash_username(username)
    payload = {
        'username': username,
        'role': role,
        'exp': datetime.datetime.utcnow() +
datetime.timedelta(minutes=30),
        'hash': hashed_username
    }
    return jwt.encode(payload, app.config['SECRET_KEY'],
algorithm="HS256")

# Token-required decorator
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')
        if not token:
            abort(403, 'Token is missing!')

        try:
            data = jwt.decode(token, app.config['SECRET_KEY'],
algorithms=["HS256"])
```

```
            # Verify the hash
            if data.get('hash') != hash_username(data['username']):
                abort(403, 'Invalid token hash')
        except jwt.ExpiredSignatureError:
            abort(403, 'Token has expired')
        except jwt.InvalidTokenError:
            abort(403, 'Invalid token')

        return f(data, *args, **kwargs)
    return decorated

# Role-based access decorator
def role_required(role):
    def wrapper(f):
        @wraps(f)
        def decorated(data, *args, **kwargs):
            if data['role'] != role:
                abort(403, 'Unauthorized access')
            return f(*args, **kwargs)
        return decorated
    return wrapper

# Login endpoint for token generation
@app.route('/login', methods=['POST'])
```

```
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')
    user = users.get(username)

    if user and user['password'] == password:
        token = generate_token(username, user['role'])
        return jsonify({"hashed token": token})

    return jsonify({"message": "Invalid credentials"}), 401

# Secured endpoint with role-based access
@app.route('/admin', methods=['GET'])
@token_required
@role_required('admin')
def admin_route():
    return jsonify({"message": "Welcome, Admin!"})

if __name__ == '__main__':
    app.run(debug=True)
```

Encryption:

```
import hashlib
import logging
import ssl
import jwt
import time
from collections import defaultdict
from datetime import datetime, timedelta
from cryptography.fernet import Fernet
import os

class RateLimiter:
    def _init_(self, max_requests, time_window):
        self.max_requests = max_requests
        self.time_window = time_window  # in seconds
        self.requests = defaultdict(list)

    def is_allowed(self, client_id):
        """Check if request is allowed based on rate limiting"""
        current_time = time.time()

        # Remove old requests outside the time window
        self.requests[client_id] = [
            req_time for req_time in self.requests[client_id]
            if current_time - req_time <= self.time_window
        ]
        # Check if client has exceeded max requests
        if len(self.requests[client_id]) >= self.max_requests:
            return False

        self.requests[client_id].append(current_time)
        return True

class IoTSecurityFramework:
    def _init_(self):
        self.key = Fernet.generate_key()
        self.cipher_suite = Fernet(self.key)
        self.setup_logging()
        # Initialize rate limiters for different services
        self.auth_limiter = RateLimiter(max_requests=3,
time_window=40)  # 5 requests per minute
        self.data_limiter = RateLimiter(max_requests=3,
time_window=40)  # 30 requests per minute
        self.connection_tracker = defaultdict(int)
        self.blacklist = set()
        self.dos_threshold = 100  # Maximum failed attempts before
blacklisting

    def setup_logging(self):
        """Configure logging for security events"""
        logging.basicConfig(
            filename='iot_security.log',
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s - %(message)s'
        )

    def encrypt_data(self, data):
        """Encrypt sensitive data before transmission"""
        try:
            encrypted_data = self.cipher_suite.encrypt(str(data).encode())
            logging.info("Data encrypted successfully")
            return encrypted_data
        except Exception as e:
            logging.error(f"Encryption failed: {str(e)}")
            return None

    def decrypt_data(self, encrypted_data):
        """Decrypt received data"""
        try:
            decrypted_data = self.cipher_suite.decrypt(encrypted_data)
            logging.info("Data decrypted successfully")
            return decrypted_data.decode()
        except Exception as e:
            logging.error(f"Decryption failed: {str(e)}")
            return None
    def generate_device_token(self, device_id, secret_key):
        """Generate JWT token for device authentication"""
        try:
            payload = {
                'device_id': device_id,
                'exp': datetime.utcnow() + timedelta(days=1)
            }
            token = jwt.encode(payload, secret_key, algorithm='HS256')
            logging.info(f"Token generated for device: {device_id}")
            return token
        except Exception as e:
            logging.error(f"Token generation failed: {str(e)}")
            return None

    def verify_device_token(self, token, secret_key):
        """Verify device token for authentication"""
        try:
            payload = jwt.decode(token, secret_key, algorithms=['HS256'])
            logging.info(f"Token verified for device:
{payload['device_id']}")
            return True
        except jwt.ExpiredSignatureError:
            logging.warning("Token has expired")
            return False
        except jwt.InvalidTokenError:
            logging.warning("Invalid token")
            return False

    def hash_password(self, password):
        """Hash passwords for secure storage"""
        return hashlib.sha256(password.encode()).hexdigest()

    def check_dos_attack(self, client_id, client_ip):
        """Check for potential DoS attack and handle accordingly"""
        try:
            if client_ip in self.blacklist:
                logging.warning(f"Blocked request from blacklisted IP:
{client_ip}")
                return False

            if not self.auth_limiter.is_allowed(client_id):
                self.connection_tracker[client_ip] += 1
                logging.warning(f"Rate limit exceeded for client: {client_id}")

                if self.connection_tracker[client_ip] > self.dos_threshold:
                    self.blacklist.add(client_ip)
                    logging.error(f"DoS attack detected! IP {client_ip}
blacklisted")
                    return False

                return False

            return True

        except Exception as e:
            logging.error(f"DoS check failed: {str(e)}")
            return False

    def handle_request(self, client_id, client_ip, request_type='data'):
        """Handle incoming requests with DoS protection"""
        try:
            # Check for DoS attack
            if not self.check_dos_attack(client_id, client_ip):
                return False, "Request blocked due to security policies"

            # Apply appropriate rate limiter based on request type
            limiter = self.auth_limiter if request_type == 'auth' else
self.data_limiter

            if not limiter.is_allowed(client_id):
```

```
            logging.warning(f"Rate limit exceeded for {request_type} 
request from {client_id}")
            return False, "Rate limit exceeded"

          return True, "Request accepted"

        except Exception as e:
          logging.error(f"Request handling failed: {str(e)}")
          return False, "Internal security error"

    def reset_client_tracking(self, client_ip):
        """Reset tracking for a client IP"""
        if client_ip in self.connection_tracker:
          del self.connection_tracker[client_ip]
        if client_ip in self.blacklist:
          self.blacklist.remove(client_ip)
        logging.info(f"Reset tracking for IP: {client_ip}")


class IoTSecurityApp:
    def _init_(self):
        self.security = IoTSecurityFramework()
        self.devices = {}  # Store registered devices
        self.current_device = None
        self.secret_key = "your-secret-key-here"

    def clear_screen(self):
        os.system('cls' if os.name == 'nt' else 'clear')

    def display_menu(self):
        self.clear_screen()
        print("\n=== IoT Device Security Framework ===")
        print("1. Register New Device")
        print("2. Login Device")
        print("3. Send Encrypted Data")
        print("4. View Security Logs")
        print("5. Check Device Status")
        print("6. Reset Security Tracking")
        print("7. Exit")
        print("=====================================")

    def register_device(self):
        print("\n=== Device Registration ===")
        device_id = input("Enter device ID: ")
        password = input("Enter device password: ")
        ip_address = input("Enter device IP address: ")

        # Hash the password and store device details
        hashed_password = self.security.hash_password(password)
        self.devices[device_id] = {
            'password': hashed_password,
            'ip_address': ip_address,
            'token': None
        }
        print("\nDevice registered successfully!")
        input("Press Enter to continue...")

    def login_device(self):
        print("\n=== Device Login ===")
        device_id = input("Enter device ID: ")
        password = input("Enter device password: ")

        if device_id in self.devices:
            if self.devices[device_id]['password'] ==
self.security.hash_password(password):
                # Generate new token
                token = self.security.generate_device_token(device_id,
self.secret_key)
                self.devices[device_id]['token'] = token
                self.current_device = device_id
                print("\nLogin successful!")
            else:
```

```
            print("\nInvalid password!")
        else:
            print("\nDevice not found!")
        input("Press Enter to continue...")

    def send_data(self):
        if not self.current_device:
            print("\nPlease login first!")
            input("Press Enter to continue...")
            return

        print("\n=== Send Encrypted Data ===")
        data = input("Enter data to encrypt and send: ")

        # Check for DoS and rate limiting
        is_allowed, message = self.security.handle_request(
            self.current_device,
            self.devices[self.current_device]['ip_address'],
            'data'
        )

        if is_allowed:
            encrypted_data = self.security.encrypt_data(data)
            print("\nData encrypted and sent successfully!")
            print(f"Encrypted data: {encrypted_data}")

            # Simulate receiving and decrypting
            decrypted_data = self.security.decrypt_data(encrypted_data)
            print(f"Decrypted data: {decrypted_data}")
        else:
            print(f"\nRequest blocked: {message}")

        input("Press Enter to continue...")

    def view_logs(self):
        print("\n=== Security Logs ===")
        try:
            with open('iot_security.log', 'r') as log_file:
            logs = log_file.readlines()
            for log in logs[-10:]:  # Show last 10 logs
                print(log.strip())
        except FileNotFoundError:
            print("No logs found.")
        input("Press Enter to continue...")

    def check_status(self):
        if not self.current_device:
            print("\nNo device logged in!")
        else:
            print(f"\nCurrent device: {self.current_device}")
            print(f"IP Address:
{self.devices[self.current_device]['ip_address']}")
            print("Token status: ", "Active" if
self.devices[self.current_device]['token'] else "Inactive")
        input("Press Enter to continue...")

    def reset_tracking(self):
        print("\n=== Reset Security Tracking ===")
        ip_address = input("Enter IP address to reset: ")
        self.security.reset_client_tracking(ip_address)
        print(f"\nSecurity tracking reset for IP: {ip_address}")
        input("Press Enter to continue...")

    def run(self):
        while True:
            self.display_menu()
            choice = input("\nEnter your choice (1-7): ")

            if choice == '1':
                self.register_device()
            elif choice == '2':
                self.login_device()
            elif choice == '3':
                self.send_data()
            elif choice == '4':
                self.view_logs()
            elif choice == '5':
                self.check_status()
            elif choice == '6':
                self.reset_tracking()
            elif choice == '7':
                print("\nExiting application...")
                break
            else:
                print("\nInvalid choice!")
                input("Press Enter to continue...")


if __name__ == "__main__":
    app = IoTSecurityApp()
    app.run()
```

Flask:

```
from flask import Flask, render_template, request, jsonify, session, redirect, url_for
from iot import IoTSecurityFramework, IoTSecurityApp
import secrets
```

```
app = Flask(__name__)
app.secret_key = secrets.token_hex(16)
security_app = IoTSecurityApp()

@app.route('/')
def index():
    if 'device_id' in session:
        return redirect(url_for('dashboard'))
    return redirect(url_for('login'))

@app.route('/register')
def register():
    if 'device_id' in session:
        return redirect(url_for('dashboard'))
    return render_template('index.html', page='register')

@app.route('/login')
def login():
    if 'device_id' in session:
        return redirect(url_for('dashboard'))
    return render_template('index.html', page='login')

@app.route('/dashboard')
def dashboard():
    if 'device_id' not in session:
        return redirect(url_for('login'))
    return render_template('index.html', page='dashboard')

@app.route('/api/register', methods=['POST'])
def api_register():
    data = request.json
    device_id = data.get('deviceId')
    password = data.get('password')
    ip_address = data.get('ipAddress')

    if device_id in security_app.devices:
        return jsonify({'success': False, 'message': 'Device ID already
exists'})

    security_app.devices[device_id] = {
        'password': security_app.security.hash_password(password),
        'ip_address': ip_address,
        'token': None
    }
    return jsonify({'success': True, 'message': 'Device registered
successfully'})

@app.route('/api/login', methods=['POST'])
```

```
def api_login():
    data = request.json
    device_id = data.get('deviceId')
    password = data.get('password')

    if device_id in security_app.devices:
        if security_app.devices[device_id]['password'] ==
security_app.security.hash_password(password):
            token =
security_app.security.generate_device_token(device_id,
security_app.secret_key)
            security_app.devices[device_id]['token'] = token
            security_app.current_device = device_id
            session['device_id'] = device_id
            return jsonify({
                'success': True,
                'message': 'Login successful',
                'deviceId': device_id,
                'ipAddress': security_app.devices[device_id]['ip_address']
            })

    return jsonify({'success': False, 'message': 'Invalid credentials'})

@app.route('/api/device-details')
def api_device_details():
    if 'device_id' not in session:
        return jsonify({'success': False, 'message': 'Not logged in'})

    device_id = session['device_id']
    device = security_app.devices.get(device_id)

    if device:
        return jsonify({
            'success': True,
            'deviceId': device_id,
            'ipAddress': device['ip_address']
        })

    return jsonify({'success': False, 'message': 'Device not found'})

@app.route('/api/logout', methods=['POST'])
def api_logout():
    session.pop('device_id', None)
    return jsonify({'success': True, 'message': 'Logged out
successfully'})

@app.route('/api/send-data', methods=['POST'])
def api_send_data():
    if 'device_id' not in session:
        return jsonify({'success': False, 'message': 'Not logged in'})

    data = request.json.get('message')
    device_id = session['device_id']

    is_allowed, message = security_app.security.handle_request(
        device_id,
        security_app.devices[device_id]['ip_address'],
        'data'
    )

    if is_allowed:
        encrypted_data = security_app.security.encrypt_data(data)
        decrypted_data = 
security_app.security.decrypt_data(encrypted_data)
        return jsonify({
            'success': True,
            'message': 'Data sent successfully',
            'encrypted': encrypted_data.decode(),
            'decrypted': decrypted_data
        })

    return jsonify({'success': False, 'message': message})
```

```
@app.route('/api/logs')
def api_logs():
    try:
        with open('iot_security.log', 'r') as log_file:
            logs = log_file.readlines()[-10:]
        return jsonify({'success': True, 'logs': logs})
    except FileNotFoundError:
        return jsonify({'success': False, 'message': 'No logs found'})


if __name__ == '_main_':
    app.run(debug=True)
```

Design:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>IoT Security Framework</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css')
}}">
    <link
href="https://fonts.googleapis.com/css2?family=Space+Mono&displa
y=swap" rel="stylesheet">
</head>
```

```
<body>
   <nav>
      <div class="logo">IoT Security framework</div>
      <div class="nav-links">
         <a href="{{ url_for('register') }}" class="nav-
register">Register</a>
         <a href="{{ url_for('login') }}" class="nav-login">Login</a>
      </div>
   </nav>

   <!-- Register Page -->
   <div class="page {% if page == 'register' %}active{% endif %}"
id="register-page">
      <h1>REGISTER A NEW DEVICE</h1>
      <form id="register-form" class="auth-form">
         <input type="text" placeholder="enter device id" required>
         <input type="password" placeholder="enter password"
required>
         <input type="text" placeholder="enter ip address" required>
         <button type="submit">add device</button>
      </form>
   </div>

   <!-- Login Page -->
   <div class="page {% if page == 'login' %}active{% endif %}"
id="login-page">
      <h1>Welcome back!</h1>
      <form id="login-form" class="auth-form">
         <input type="text" placeholder="enter device id" required>
         <input type="password" placeholder="enter password"
required>
         <button type="submit">Sign in</button>
      </form>
   </div>

   <!-- Dashboard Page -->
   <div class="page {% if page == 'dashboard' %}active{% endif %}"
id="dashboard-page">
      <div class="dashboard-container">
         <div class="left-panel">
            <div class="data-transfer-card">
               <h2>Data transfer</h2>
               <input type="text" placeholder="enter message"
id="message-input">
               <div class="message-output"></div>
               <button class="send-btn">send</button>
            </div>
            <div class="logs-card">
               <h2>SECURITY LOGS</h2>
```

```
                <div class="logs-content">
                    <!-- Logs will be inserted here -->
                </div>
                <button class="refresh-btn">refresh</button>
            </div>
        </div>
        <div class="user-details-card">
            <div class="user-icon">👤</div>
            <h2>user details</h2>
            <p class="device-id">Device ID: <span id="device-id-
display">--</span></p>
            <p class="ip-address">IP: <span id="ip-address-display">--
</span></p>
            <button class="logout-btn">logout</button>
        </div>
    </div>
</div>

<!-- Notification Modal -->
<div id="notification-modal" class="modal">
    <div class="modal-content">
        <h2 class="modal-title">ERROR!</h2>
        <p class="modal-message">Invalid credentials</p>
        <button class="modal-close">close</button>
    </div>
</div>

<script src="{{ url_for('static', filename='script.js') }}"></script>
</body>
</html>
```

CSS:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Space Mono', monospace;
}

body {
    min-height: 100vh;
    background: linear-gradient(45deg, #e6eeff, #ffebf2);
}

nav {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 1rem 2rem;
    background: #000;
    color: white;
}

.nav-links a {
    color: white;
    text-decoration: none;
    margin-left: 2rem;
    transition: opacity 0.2s;
}

.nav-links a:hover {
    opacity: 0.8;
}

.page {
    display: none;
    flex-direction: column;
    align-items: center;
    padding: 4rem 2rem;
    min-height: calc(100vh - 60px);
}
```

```
.page.active {
    display: flex;
}

h1 {
    font-size: 2.5rem;
    margin-bottom: 3rem;
}

h2 {
    margin-bottom: 1.5rem;
}

.auth-form {
    display: flex;
    flex-direction: column;
    gap: 1.5rem;
    width: 100%;
    max-width: 500px;
}

input {
    padding: 1rem;
    border: none;
    border-radius: 25px;
    background: white;
    font-size: 1rem;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}

button {
    padding: 1rem 2rem;
    border: none;
    border-radius: 25px;
    background: #000;
    color: white;
    font-size: 1rem;
    cursor: pointer;
    transition: transform 0.2s;
}

button:hover {
    transform: scale(1.02);
}

.dashboard-container {
    display: flex;
    gap: 2rem;
    width: 100%;
    max-width: 1200px;
}

.left-panel {
    flex: 1;
    display: flex;
    flex-direction: column;
    gap: 2rem;
}

.data-transfer-card {
    background: rgba(255, 235, 242, 0.5);
    padding: 2rem;
    border-radius: 15px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}

.message-output {
    height: 100px;
    background: white;
    border-radius: 10px;
    margin: 1rem 0;
```

```
  padding: 1rem;
  font-family: monospace;
}

.logs-card {
  background: rgba(230, 238, 255, 0.5);
  padding: 2rem;
  border-radius: 15px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  display: flex;
  flex-direction: column;
}

.logs-content {
  background: #333;
  color: #fff;
  padding: 1rem;
  border-radius: 10px;
  font-family: monospace;
  margin-top: 1rem;
  height: 300px;
  overflow-y: auto;
  white-space: pre-wrap;
  font-size: 0.9rem;
```

```
   line-height: 1.4;
}

.refresh-btn {
   margin-top: 1rem;
   width: 100%;
   background: rgba(0, 0, 0, 0.8);
}

.user-details-card {
   background: rgba(230, 238, 255, 0.5);
   padding: 2rem;
   border-radius: 15px;
   width: 300px;
   text-align: center;
   box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
   height: fit-content;
}

.user-icon {
   font-size: 3rem;
   margin-bottom: 1rem;
}
```

```
.device-id,
.ip-address {
   margin: 0.5rem 0;
}

.logout-btn {
   margin-top: 1rem;
   width: 100%;
}

/* Modal Styles */
.modal {
   display: none;
   position: fixed;
   top: 0;
   left: 0;
   width: 100%;
   height: 100%;
   background-color: rgba(0, 0, 0, 0.5);
   z-index: 1000;
   display: flex;
   justify-content: center;
   align-items: center;
   opacity: 0;
```

```
    pointer-events: none;
    transition: opacity 0.3s ease;
}

.modal.active {
    opacity: 1;
    pointer-events: auto;
}

.modal-content {
    background: white;
    padding: 2rem;
    border-radius: 15px;
    width: 90%;
    max-width: 500px;
    text-align: center;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    transform: translateY(-20px);
    transition: transform 0.3s ease;
}

.modal.active .modal-content {
    transform: translateY(0);
}
```

```
.modal-title {
   font-size: 2rem;
   margin-bottom: 1.5rem;
}

.modal-message {
   font-size: 1.5rem;
   margin-bottom: 2rem;
}

.modal-close {
   background: #000;
   color: white;
   border: none;
   padding: 0.8rem 2rem;
   border-radius: 25px;
   font-size: 1rem;
   cursor: pointer;
   transition: transform 0.2s;
}

.modal-close:hover {
   transform: scale(1.05);
}

/* Responsive Design */
@media (max-width: 768px) {
   .dashboard-container {
      flex-direction: column;
   }

   .user-details-card {
      width: 100%;
   }

   h1 {
      font-size: 2rem;
      text-align:center;
   }
}
```

JS:

```
function showNotification(title, message, isError = false) {
   const modal = document.getElementById('notification-modal');
   const modalTitle = modal.querySelector('.modal-title');
   const modalMessage = modal.querySelector('.modal-message');
```

```
    modalTitle.textContent = title;
    modalMessage.textContent = message;

    modal.classList.add('active');
}

function closeNotification() {
    const modal = document.getElementById('notification-modal');
    modal.classList.remove('active');
}

document.addEventListener('DOMContentLoaded', function() {
    // Add modal close button listener
    const modalClose = document.querySelector('.modal-close');
    if (modalClose) {
        modalClose.addEventListener('click', closeNotification);
    }

    // Register form handler
    const registerForm = document.getElementById('register-form');
    if (registerForm) {
        registerForm.addEventListener('submit', async (e) => {
            e.preventDefault();
            const inputs = registerForm.getElementsByTagName('input');
    }

    const response = await fetch('/api/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        deviceId: inputs[0].value,
        password: inputs[1].value,
        ipAddress: inputs[2].value
      })
    });
    const data = await response.json();

    if (data.success) {
      showNotification('SUCCESS!', data.message);
      setTimeout(() => {
        window.location.href = '/login';
      }, 2000);
    } else {
      showNotification('ERROR!', data.message);
    }
  });
}
```

```
    // Login form handler
    const loginForm = document.getElementById('login-form');
    if (loginForm) {
        loginForm.addEventListener('submit', async (e) => {
            e.preventDefault();
            const inputs = loginForm.getElementsByTagName('input');
            const response = await fetch('/api/login', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({
                    deviceId: inputs[0].value,
                    password: inputs[1].value
                })
            });
            const data = await response.json();

            if (data.success) {
                document.getElementById('device-id-display').textContent =
data.deviceId;
                document.getElementById('ip-address-display').textContent
= data.ipAddress;
                showNotification('SUCCESS!', data.message);
                setTimeout(() => {
                    window.location.href = '/dashboard';
                }, 2000);
            } else {
                showNotification('ERROR!', data.message);
            }
        });
    }

    // Dashboard functionality
    if (document.getElementById('dashboard-page')) {
        // Get device details on page load
        async function updateDeviceDetails() {
            const response = await fetch('/api/device-details');
            const data = await response.json();
            if (data.success) {
                document.getElementById('device-id-display').textContent =
data.deviceId;
                document.getElementById('ip-address-display').textContent
= data.ipAddress;
            }
        }

        // Initial device details update
```

```
        updateDeviceDetails();

        const sendBtn = document.querySelector('.send-btn');
        const messageInput = document.getElementById('message-
input');
        const messageOutput = document.querySelector('.message-
output');

        sendBtn.addEventListener('click', async () => {
            const message = messageInput.value;
            if (!message) {
                showNotification('ERROR!', 'Please enter a message');
                return;
            }

            const response = await fetch('/api/send-data', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ message })
            });
            const data = await response.json();
```

```
  if (data.success) {
    messageOutput.innerHTML = `
      Encrypted: ${data.encrypted}
      <br><br>
      Decrypted: ${data.decrypted}
    `;
    messageInput.value = '';
    showNotification('SUCCESS!', data.message);
  } else {
    showNotification('ERROR!', data.message);
  }
});

// Logout handler
const logoutBtn = document.querySelector('.logout-btn');
logoutBtn.addEventListener('click', async () => {
  const response = await fetch('/api/logout', { method: 'POST' });
  const data = await response.json();
  showNotification('SUCCESS!', data.message);
  setTimeout(() => {
    window.location.href = '/login';
  }, 2000);
});
```

```
        // Update logs function
        async function updateLogs() {
            const response = await fetch('/api/logs');
            const data = await response.json();
            if (data.success) {
                document.querySelector('.logs-content').innerHTML = data.logs.join('');
            }
        }

        // Refresh button handler
        const refreshBtn = document.querySelector('.refresh-btn');
        refreshBtn.addEventListener('click', updateLogs);

        // Initial logs update and set interval
        updateLogs();
        setInterval(updateLogs, 5000);
    }
});
```