Automated Testing Report

Project Name: User Registration and Login System

Date: 2025/12/3

Tester:

SWE2309509 He Bolin

SWE2309520 Li Ruibing

SWE2309540 Wu Yi

1. Test Summary

Objective

To validate the end-to-end functionality and data integrity of user registration and login modules. The primary goal is to ensure that:

1） Core Functionality: Valid users can register and login successfully without errors.

2） Security: Security mechanisms such as account lockout and token validation work robustly to protect user account and sensitive information.

3） Reliability: The system handles invalid inputs or unexpected data (e.g., extremely long emails) properly without crashing.

Tools

Automation Framework: Python (Selenium WebDriver)

Database Connector: psycopg2-binary (Python PostgreSQL Adapter)

Browser Driver: ChromeDriver

Browser Developer Tools: Chrome DevTools

IDE/Editor: Pycharm

Test Information Record: Excel

Test Environment

OS: Windows 11

Browser: Google Chrome

Application URL: http://localhost:8000

Database: PostgreSQL (Containerized via Docker: *svv_auth* DB)

2. Pre-prepared code

This section explains the preset values (such as imports) of my scripts. To make the testing process smoother, I encapsulated various basic functions, which makes the code easier to read and user.

I used full XPath for all element locators to ensure the web elements are positioned accurately.

Script framework for basic operations:

```
import psycopg2
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time


#set up the browser
def setting():
    q1=Options()
    q1.add_argument('--no-sandbox')
    q1.add_experimental_option('detach',True)
    a1=webdriver.Chrome(service=Service('chromedriver.exe'),options=q1)
    return a1


#set up database
DB_CONFIG = {
    "host": "localhost",
```

```
    "port": "5432",
    "database": "svv_auth",
    "user": "svv_user",
    "password": "svv_password_change_me"
}

#basic functions for testing
def enter_register_page(driver):
    a = driver.find_element(By.XPATH,"/html/body/div/div[1]/div/div[2]/p/a")
    a.click()

def enter_login_page(driver):
    a = driver.find_element(By.XPATH,"/html/body/div/div[1]/div/div[2]/p/a")
    a.click()

def input_username_login(driver, username):
    a = driver.find_element(By.XPATH,
"/html/body/div/div[1]/div/form/div[1]/input")
    a.clear()
    a.send_keys(username)

def input_password_login(driver, password):
    a = driver.find_element(By.XPATH,
"/html/body/div/div[1]/div/form/div[2]/input")
    a.clear()
    a.send_keys(password)

def input_username_register(driver, username):
    a = driver.find_element(By.XPATH,
```

```
"/html/body/div/div[1]/div/form/div[1]/input")
    a.clear()
    a.send_keys(username)

def input_email_register(driver, email):
    a = driver.find_element(By.XPATH,
"/html/body/div/div[1]/div/form/div[2]/input")
    a.clear()
    a.send_keys(email)

def input_password_register(driver, password):
    a = driver.find_element(By.XPATH,
"/html/body/div/div[1]/div/form/div[3]/input")
    a.clear()
    a.send_keys(password)

def input_password_confirm(driver, password):
    a = driver.find_element(By.XPATH,
"/html/body/div/div[1]/div/form/div[4]/input")
    a.clear()
    a.send_keys(password)

def click_login_button(driver):
    btn = driver.find_element(By.XPATH,
"/html/body/div/div[1]/div/form/button")
    btn.click()

def click_register_button(driver):
    btn = driver.find_element(By.XPATH,
```

```
"/html/body/div/div[1]/div/form/button")
        btn.click()


#web(d means driver)
d=setting()
d.get('http://localhost:8000')


#database
conn = psycopg2.connect(**DB_CONFIG)
cursor = conn.cursor()


try:
#this is where I put the operations codes to be executed according to the test cases


except Exception as e:
        print(f"\nError: {e}")


finally:
        print("\nTesting Complete.")
```

Another script framework for security and reliability testing:

```
import sys
import os
import time
import statistics
import random
from pathlib import Path
```

```
import bcrypt


# Hack to fix passlib + bcrypt 4.x issue
if not hasattr(bcrypt, '__about__'):
    try:
        from collections import namedtuple


        Version = namedtuple('Version', ['__version__'])
        bcrypt.__about__ = Version(bcrypt.__version__)
    except Exception:
        pass


sys.path.append(os.getcwd())


from fastapi import FastAPI
from fastapi.testclient import TestClient
from sqlalchemy import create_engine, text
from sqlalchemy.orm import sessionmaker
from sqlalchemy.pool import StaticPool
from jose import jwt


from backend.api import router, failed_login_attempts
from backend.database import Base, get_db
from backend.models import User
from backend.auth import get_password_hash
from backend.config import settings



class Logger(object):
```

```
    def __init__(self, filename="system_verification.log"):
        self.terminal = sys.stdout
        self.log = open(filename, "w", encoding='utf-8')

    def write(self, message):
        self.terminal.write(message)
        self.log.write(message)
        self.log.flush()

    def flush(self):
        self.terminal.flush()
        self.log.flush()



# --- Setup ---
app = FastAPI()
app.include_router(router)

SQLALCHEMY_DATABASE_URL = "sqlite:///:memory:"
engine = create_engine(
    SQLALCHEMY_DATABASE_URL,
    connect_args={"check_same_thread": False},
    poolclass=StaticPool,
)
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)



def override_get_db():
```

```
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()



app.dependency_overrides[get_db] = override_get_db
Base.metadata.create_all(bind=engine)
client = TestClient(app, raise_server_exceptions=False)


if __name__ == "__main__":
    # Redirect stdout to log file
    sys.stdout = Logger()

#Here I also encapsulated various functions for security and reliability tests, but it' s too long so I will mention them later in the "Defect summary" section.
```

3. Automatic Test Cases

3.1 Functional Test Cases (REG / LGN)

3.1.1 REG Test Cases

- **Registration: Main Flow**

  Test Objective: Register success with all valid inputs and can login with created account

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_001	Register with all valid inputs	All inputs are valid	1. Fill Username, Email, Password, Confirm 2. Click Register	Registration success; redirect to Login or auto-login	**TEST PASS** Registration Successful
REG_002	Log in right after registration	All inputs are valid	1. Fill valid inputs in all fields 2. Click Register 3. Return to login page, log in immediately	Registration success; Login success	**TEST PASS** Registration Successful

- **Registration: Mandatory Field Validation**

  Test Objective: Register Failure with invalid or empty input in some/all fields

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_010	Register with Empty Username	Username= (empty); Other inputs valid	1. Leave Username empty 2. Fill other fields with valid inputs 3. Click Register	Show “Username is required” error	**TEST PASS** Show “Username must be at least 3 characters”
REG_011	Register with Empty Email	Email= (empty); Other inputs valid	1. Leave Email empty 2. Fill other fields with valid inputs 3. Click Register	Show “Email is required” error	**TEST PASS** Show “Please enter a valid email address” error
REG_012	Register with Empty Password	Password= (empty); Confirm= (empty) Other inputs valid	1. Leave Password (and confirm password) empty 2. Fill other fields with valid input 3. Click Register	Show 'Password is required' error	**TEST PASS** Show “Password must be at least 8 characters” error
REG_013	Register with Empty Confirm Password	Confirm= (empty) Other inputs valid	1. Leave Confirm empty 2. Click Register	Show 'Please confirm your password' error	Meaning of warning is not clear Show “Password must be at least 8 characters” error

REG_014	Register with invalid inputs in all fields	Username= (empty); Email= 1; Password= 123; Confirm= 456	1. Leave Username empty 2. Enter other invalid inputs 3. Click Register	All the invalid inputs should be highlighted	Warning is not adequate Only “Invalid Email” is highlighted

- **Registration: Process Validation**

  Test Objectives: Verify register process robustness

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_020	Page refresh before registering	All inputs valid	1. Fill valid inputs in all fields 2. Refresh the page before clicking on ‘Register’	All inputs should be cleared after refreshing	**TEST PASS** As expected
REG_021	Page refresh while registering	All inputs valid	1. Fill valid inputs in all fields 2. Use scripts to enable clicking ‘Register’ and refreshing page at the same time	The account should be set up correctly after refreshing is over	**TEST PASS** Account set up successfully

- **Registration: Email Input Validation**

  Test Objectives: Register failure with invalid-format/ out-of-bound email

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_030	Register with invalid email format with missing @	Email=abc.com	1. Enter invalid email 2. Click Register	Show “Invalid email format” error	**TEST PASS** Show “Please add a ‘@’. ‘abc.com’ does not have a ‘@’.”
REG_031	Register with invalid email format with missing domain	Email=abc@	1. Enter invalid email 2. Click Register	Show “Invalid email format” error	**TEST PASS** Show “Please enter a part following ‘@’. ‘abc@’ is incomplete.”
REG_032	Register email with plus-addressing	Email= user+tag@example.com	1. Enter plus-addressed email 2. Click Register	Should accept as valid format	**TEST PASS** Show “Registration successful”; Jump back to login page.
REG_033	Register email case-insensitive uniqueness	Email= EXIST@example.com	1. Enter email with different case 2. Click Register	Show “Username already exists” error (Treat as same email)	**TEST PASS** Show “Registration failed: Email already registered”

REG_034	Register email length boundary test	Email= (>255 chars)@test.com	1. Enter a prepared extremely long email 2. If the system did not warn us, continue to add more characters 3. Click Register	System sanitizes input; Script does NOT execute	No any email length constraints System page crashes

- **Password Input Validation**

  Test Objectives:

  - 1) Failure to comply with complexity rules (Upper, Lower, Digit, Special Char, whitespace-only) will cause register failure.
  - 2) Length boundary tests (MIN boundary value: 8; MAX boundary value: 32)

BVA – Password Length		
Invalid (min -1)	**Valid** (min, min +1, max -1, max)	**Invalid** (max +1)
7	**8, 9, 127, 128**	**129**

- 3) Visual inline hints verification (UX)

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_040	Register password lacks uppercase	Password=abc123!@; Confirm=abc123!@	1. Enter password without uppercase 2. Click Register	Show “Password must contain at least one uppercase letter” error	**TEST PASS** As expect
REG_041	Register password lacks lowercase	Password=ABC123!@; Confirm=ABC123!@	1. Enter password without lowercase 2. Click Register	Show “Password must contain at least one lowercase letter” error	**TEST PASS** As expect
REG_042	Register password lacks digit	Password=Abcdef!@; Confirm=Abcdef!@	1. Enter password without digit 2. Click Register	Show “Password must contain at least one digit” error	**TEST PASS** As expect
REG_043	Register password lacks special character	Password=Abc12345; Confirm=Abc12345	1. Enter password without special char 2. Click Register	Show “Password must contain at least one special character” error	**TEST PASS** As expect
REG_044	Register with whitespace-only password	Password= ‘ ’	1. Enter spaces for passwords 2. Click Register	Reject as invalid; show error	**TEST PASS** Show “Password contains illegal characters.”
REG_045		*BV 1:* Enter length 7		Invalid (Rejected)	**TEST PASS**

	Register password length boundary Value test cases	*BV 2:* Enter length 8	1. Enter passwords with *BV 1* / *BV 2* / *BV 3* / *BV 4* / *BV 5* / *BV 6* 2. Click Register	Valid (Accepted)	As expect
		BV 3: Enter length 9		Valid (Accepted)	
		BV 4: Enter length 127		Valid (Accepted)	
		BV 5: Enter length 128		Valid (Accepted)	
		BV 6: Enter length 129		Invalid (Rejected)	
REG_046	Register when confirm password not match with password	Inspect UI	1. Focus password field 2. Observe hints	Password rules displayed and update as user types	1. No real-time password UI inspector at first time register; 2. No password up limitation hint before submit.

- **Registration: Duplicate Registration**

  ***Test Objectives**:* Register failure when username or email already exists

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_050	Register when username already exists	Register with username from an existing account	1. Enter valid data in all fields with an existing username 2. Click Register	Show “Username already exist” error	**TEST PASS** As expect

REG_051	Register when email already exists	Register with email from an existing account	1. Enter valid data in all fields with an existing email 2. Click Register	Show “Email already exist” error	**TEST PASS** Show “Passwords do not match” error
REG_052	Register two accounts with same inputs at the same time	Prepare two sets of registration data with their username, email, and password are all the same	1. Use script to enable filling information and register at the same time 2. Check database	Only on account is successfully set up	**TEST PASS** As expect

- **Username Input Validation**

  Test Objectives:

  - 1) System auto trim spaces if register with leading/trailing spaces in username
  - 2) Length boundary tests (MIN boundary value: 3; MAX boundary value: 50)

BVA – Username Length		
Invalid (min -1)	**Valid** (min, min +1, max -1, max)	**Invalid** (max +1)
2	**3, 4, 49, 50**	**51**

- 3) Internationalization support (Unicode/Chinese characters)

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_060	Register Username has Leading/trailing spaces	Username= ‘ newuser ’	1. Enter username with spaces 2. Click Register	Trim spaces and register OR reject as invalid and show error	**TEST PASS** Show “Username can only contain letters, numbers, and underscores” error
REG_061	Register Username length boundary Value test cases	*BV 1:* Enter length 2	1. Enter username with *BV 1 / BV 2 / BV 3 / BV 4 / BV 5 / BV 6* 2. Click Register	Invalid (Rejected)	**TEST PASS (BV)** No “Username must be at most 50 characters” hint right after the user enters the username (but after submit)
		BV 2: Enter length 3		Valid (Accepted)	
		BV 3: Enter length 4		Valid (Accepted)	
		BV 4: Enter length 49		Valid (Accepted)	
		BV 5: Enter length 50		Valid (Accepted)	
		BV 6: Enter length 51		Invalid (Rejected)	
REG_062	Register with common special chars in username (.-_)	Username= ‘john.doe_jr-1’	1. Enter username with . _ - 2. Click Register	Accept or reject based on spec; should not crash	**TEST PASS** Show “Username can only contain

					letters, numbers, and underscores” error
REG_063	Register with Unicode char in username	Username= ‘hh←’	1. Enter Unicode username 2. Click Register	Accept or reject based on spec; must not crash	**TEST PASS** Show “Username can only contain letters, numbers, and underscores” error

- **Registration: Malicious Injection**

  Test Objectives: XSS (Cross-Site Scripting) must be prevented

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_070	XSS attempt in register username or email	Username= <script>alert(1)</script>	1. Enter malicious payload 2. Click Register	System sanitizes input; Script does NOT execute	**TEST PASS** Show “Username can only contain letters, numbers, and underscores” error

- **Registration: Network Condition**

  Test Objectives: Correct network timeout handling, graceful failure on slow connections

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
REG_9.1	Register with slow network	Valid data	1. Use Browser tool (F12) to control network interrupt 2. Recover network after 1 min	Show loading indicator and friendly timeout/error message Or register successfully after network recover	**TEST PASS** Registration Successful after network recovered

3.1.2 LGN Test Cases

- **Login: Main Flow**

  Test Objective: Login/Logout success with all valid credentials, and navigation link to Register page works

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
LGN_001	Login with valid credentials	Username=testuser; Password=Password1!	1. Enter Username 2. Enter Password 3. Click Login	Login successful; redirect to Dashboard	**TEST PASS** As expect

LGO_001	Logout after enter dashboard page	Inspect UI after click “Log out” button	1. Click Logout button	Logout and navigate to login page.	**TEST PASS** As expect
LGN_002	Login link to Register navigates correctly	Inspect UI after click “Register now”	1. Click Register link	Page navigates to Register page	**TEST PASS** As expect

- **Login: Login Input Validation**

  Test Objective: Login failure with invalid input or edge cases

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
LGN_010	Login with username and password empty	Username=(empty), Password=(empty)	1. Click “Login” button directly	Error “Username is required”; Error “Password is required”; stay on Login page	**TEST PASS** As expect

LGN_011	Login with username empty	Username= (empty); Password=Password1!	1. Leave Username empty 2. Enter Password 3. Click Login	Error “Username is required”; stay on Login page	**TEST PASS** As expect
LGN_012	Login with password empty	Username=testuser; Password= (empty)	1. Enter Username 2. Leave Password empty 3. Click Login	Error “Password is required”; stay on Login page	**TEST PASS** As expect
LGN_013	Login with non-existent username	Username=no_user; Password=AnyPass1!	1. Enter unknown Username 2. Enter Password 3. Click Login	Error “User not found” or generic auth error	**TEST PASS** Show “Login failed: Incorrect username or password”; Show Warning: Multiple failed attempts may lock your account temporarily
LGN_014	Login with incorrect password	Username=testuser; Password=WrongPass1!	1. Enter Username 2. Enter wrong Password 3. Click Login	Error “Incorrect password” and deny access	**TEST PASS** Same as above

LGN_015	Login with leading/trailing spaces in username	Username=' testuser ' ; Password=Password1!	1. Enter Username with spaces 2. Enter Password 3. Click Login	Trim spaces then login success OR show invalid username if spaces considered	**TEST PASS** Show “Login failed: Incorrect username or password”;
LGN_016	Login with case-sensitive username handling	Username=TestUser vs testuser; Password correct	1. Try with different casing 2. Click Login	Behavior matches spec (either case-sensitive or not); consistent	**TEST PASS** Same as above
LGN_017	Login with special characters in username	Username=user!@#; Password=Password1!	1. Enter Username with special chars 2. Enter Password 3. Click Login	Accept or reject according to spec; should not crash	**TEST PASS** Same as above
LGN_018	Login Username length boundary Value test cases	*BV 1:* Enter length 2	1. Enter username with *BV 1* / *BV 2* / *BV 3* / *BV 4* / *BV 5* / *BV 6* 2. Click Login	Invalid	**TEST PASS**
		BV 2: Enter length 3		Valid	
		BV 3: Enter length 4		Valid	
		BV 4: Enter length 49		Valid	
		BV 5: Enter length 50		Valid	
		BV 6: Enter length 51		Invalid	

- **Login: System Lockout Behavior**

  Test Objectives: Verify the system's resilience against abuse attempts and malicious input patterns.

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
LGN_020	Lockout and unlock when login fail	Invalid username and password	1. Enter invalid username and password 2. Click login 3. If the system warns "Login failed: Multiple failed attempts may lock your account temporarily", click login again 4. Continue to log in until the system is locked	The system is locked after several times of login failure, tell the user to wait	**TEST PASS** System is locked after five times of login failure; user is unable to login for the next 1 minutes

LGN_021	Use another IP to attempt login after one IP is locked	Invalid username and password; Two different IP: “1.1.1.1” and “2.2.2.2”	1. Enter invalid username and password 2. In one IP, click login until the system is locked 3. Switch to another IP and try login with same username and password	In the other IP, the system should still be locked and not allow further login attempts	Raised security problem Further login attempt is allowed in a new IP
LGN_022	Account lockout policy bypass	Invalid username and password; Valid username and password	1. Enter invalid username and password for four times 2. Enter valid username and password for one time 3. Logout 4. Try to enter invalid username and password again for several times	The system should lock when total number of login failure reaches 5 times	Failed Login Counter Resets After Successful Login Successful login resets the failure count to zero. Lockout requires 5 new consecutive failures post-logout, ignoring previous attempts.

3.2 Non-functional Test Cases (Database, Security)

3.2.1 Database Test Cases

- **DAT _1.x: Database Verification**

  ***Test Objectives**:*

  - 1) Verify data consistency between UI and Database for creation and immediate retrieval.
  - 2) Validate access denial and session termination logic upon user deletion.
  - 3) Ensure authentication mechanisms correctly reflect backend data updates (Deactivation & Credential changes).
  - 4) Verify database constraints regarding username case sensitivity.

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
DAT _001	Registration Data Integrity check (UI to DB)	1. Username = “user01” 2. Email= email01@test.com 3. Password = Abc123!@	1. Enter register page 2. Input valid username, email, password, confirm password 3. Click Register Button 4. Check database for user info	The user info in the database is as same as what we used in registration	**TEST PASS** As expect

DAT _002	Login After Registration	1. Username = “user02” 2. Email= email02@test.com 3. Password = Abc123!@	1. Register a new account with valid inputs 2. Back to Login Page 3. Login using the newly registered credentials 4. Click Login Button	User login successfully and is navigated to dashboard page	**TEST PASS** As expect
DAT _003	Login After user data is deleted	1. Username = “user03” 2. Email= email03@test.com 3. Password = Abc123!@	1. Register a new account and ensure logged out 2. Delete user record from DB 3. Try login with the deleted credentials	For the first time the user can login successfully, after user is deleted, the login fails	**TEST PASS** As expect
DAT _004	Delete user while user is still in dashboard page	1. Username = “user01” 2. Email= email01@test.com 3. Password = Abc123!@	1. Login with a valid account 2. Enter dashboard page 3. Check DB → delete user data and observe interface 4. Check DB → refresh the page multiple times, observe interface	System invalidates the session, log the user out, and redirect them to Login Page	Session persists after DB deletion Deleted user remains logged in; Refreshing the page continues to display user info.

DAT _005	Login after deactivate account	1. Username = “user01” 2. Email= email02@test.com 3. Password = Abc123!@ 4. New value of ‘is_active’ = False	1. Login with a valid account then manually deactivate account in DB (is_active= ‘false’) 2. Verify access is denied upon page refresh and re-login attempts 3. Reactivate account in DB (is_active= ‘true’) and confirm login works again	Logging failed; the system shows “account is banned” or “wrong username/password”	**TEST PASS** As expect
DAT _006	Login After username modification	1. Username = “user01” 2. Email= email03@test.com 3. Password = Abc123!@ 4. New value of “username” = user12345	1. Register “user01” and ensure login works 2. Update “user01” to “user12345” in DB then logout 3. Try login with “user01” 4. Try login with “user12345”	User cannot login with “user01” after modification, can successfully log in with “user12345”	**TEST PASS** As expect

DAT _007	Login After password modification	1. Username = “user01” 2. Email= email03@test.com 3. Password = Abc123!@ 4. New value of “username” = user12345	1. Register an account with password ‘Abc123!@’ 2. Modify the password to ‘123Abc!@’ in DB 3. Try login with new password 4. Try login with old password 5. Try login with previous password token	New password should success and old password fail, previous token login should fail.	Auth Broken; Token Valid Both password login failed (password is hashed), but previous token success
DAT _008	Case sensitivity chaos	1. Username1 = “user01”, email and password valid 2. Username2 = “USER01”, email and password valid	1. Register two accounts with usernames that differ in case 2. Delete ‘user01’ in DB 3. Log in with ‘user01’ then logout 4. Log in with ‘USER01’	The ‘user01’ should not be able to log in while ‘USER01’ can	**TEST PASS** As expect

3.2.2 SEC Test Cases

- **Security Validation**

  Test Objectives:

- 1) Validate input sanitization mechanisms against common injection attacks
- 2) Verify secure transport protocols to ensure credentials are not exposed in URLs or GET requests.
- 3) Ensure secure session management via JWT implementation (Headers, Storage, and Secret Keys).

TC ID	TC Name	Test Data	Steps to Execute	Expected Result	Actual Result
SEC _001	Login with SQL injection pattern	Username:' OR '1'= '1; Password=Abc123!@	1. Enter SQL injection payload in username field 2. Enter random password 3. Click Login	System rejects input or shows generic error; No database error or bypass.	**TEST PASS** As expect
SEC _002	Login with XSS injection in Username	Username: <script>alert(1)</script>; Password=Abc123!@	1. Enter script tag in username 2. Submit form 3. Check if script executes	Script tags are escaped/sanitized; No popup appears	**TEST PASS** As expect

SEC _003	Login form submit via HTTP POST	Valid/Invalid Credentials	1. Open DevTools (Network) 2. Perform Login 3. Check Request Method	Login must use HTTP POST method.	**TEST PASS** As expect
SEC _004	Presence of sensitive data in URLs after Login	Valid Credentials	1. Perform Login 2. Observe Browser Address Bar	URL must not contain query parameters like ?password=...	Session persists after DB deletion Deleted user remains logged in; Refreshing the page continues to display user info.
SEC _005	JWT token passed via Authorization header	Valid Credentials	1. Login 2. Inspect subsequent API requests	Token is sent in Authorization: Bearer <token> header.	**TEST PASS** As expect

SEC _006	Verify login response time consistency	Valid and invalid username and a invalid password	1. Enter a valid username and wrong password. 2. Measure response time (Time A) 3. Enter an invalid username. 4. Measure response time (Time B).	Time A,B should be roughly the same	**Results are different significantly** Time B is significantly faster than Time A.

Compilation of Automated Test Results

Method	Module	Total	Pass	Fail	Pass Rate
Black Box	**REG**	34	31	**3**	91.1%
	LGN	12	10	**2**	83.3%
	DAT	8	6	**2**	75.0%
	SEC	6	5	**1**	83.3%

4. Summary of Defects

4.1 Usability Defects

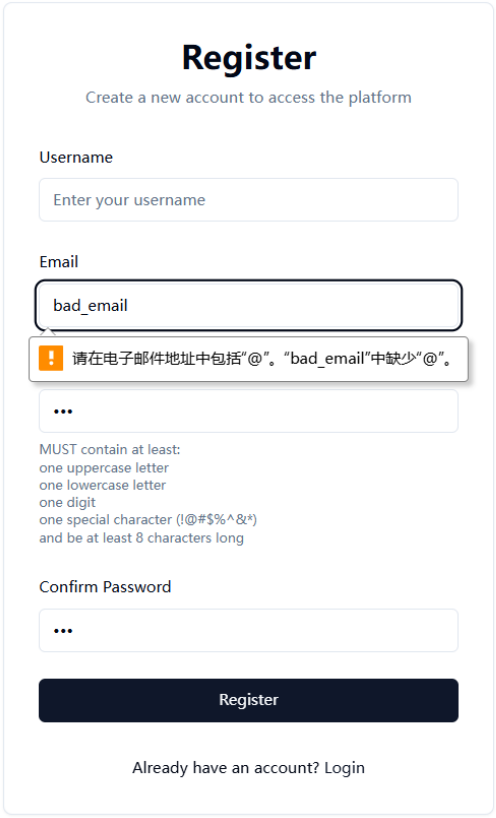
1) **DEF_01:** Registration with all inputs invalid

Description: When a user fills in all fields with invalid data (e.g., empty username, bad email, short password) and clicks Register, the system should highlight all mistakes. However, currently, it only shows an error for the Email field. The user won't know their Username or Password is also wrong.

Test Script:

```
try:
        input_username_register(d, "") (Empty username).
        input_email_register(d, "bad_email") (Invalid format).
        input_password_register(d, "123") (Too short).
        click_register_button(d).
```

Output: After running the program, only the 'Email error' is found on register page but failed to find other error messages for Username and Password.



Impact: Users will be confused because they think only the email is wrong, but actually everything is wrong.

4.2 Security Defects

1) **DEF_02: Lockout Mechanism Bypass**

Description: The system trusts the X-Forwarded-For HTTP header blindly. Whether the account should be locked is determined by IP, for example, if your account is locked in one IP, you can simply modify your IP so you can try logging in again. Attackers can easily bypass this by switching Ips.

Test Script：

```
def test_proxy_forwarding_support():
    print("\n--- [Security] Proxy Forwarding Support ---")
    failed_login_attempts.clear()
    setup_user("user_xff", "Password123!")
    ip_a = "10.0.0.1"

    # Block IP A
    for i in range(5):
        client.post(
            "/api/auth/token",
            data={"username": "user_xff", "password": "WrongPassword"},
            headers={"X-Forwarded-For": ip_a}
        )

    # Try with spoofed IP
    spoofed_ip = "10.0.0.2"
    res_spoofed = client.post(
        "/api/auth/token",
```

```
            data={"username": "user_xff", "password": "Password123!"},
            headers={"X-Forwarded-For": spoofed_ip}
        )
        log_operation("POST", "/api/auth/token", res_spoofed,
data={"username": "user_xff", "password": "Password123!"},
                      headers={"X-Forwarded-For": spoofed_ip})

        if res_spoofed.status_code == 200:
            print("Result: WARNING (Rate limit bypassed via X-Forwarded-For header)")
        else:
            print(f"Result: PASSED (Rate limit enforced correctly)")
```

Terminal execution:

```
PS C:\Users\hebolin> curl.exe -X POST "http://localhost:8000/api/auth/token" -H "X-Forwarded-For: 1.1.1.1" -d "username=admin&password=wrong"
{"detail":"Incorrect username or password"}
PS C:\Users\hebolin> curl.exe -X POST "http://localhost:8000/api/auth/token" -H "X-Forwarded-For: 1.1.1.1" -d "username=admin&password=wrong"
{"detail":"Incorrect username or password"}
PS C:\Users\hebolin> curl.exe -X POST "http://localhost:8000/api/auth/token" -H "X-Forwarded-For: 1.1.1.1" -d "username=admin&password=wrong"
{"detail":"Incorrect username or password"}
PS C:\Users\hebolin> curl.exe -X POST "http://localhost:8000/api/auth/token" -H "X-Forwarded-For: 1.1.1.1" -d "username=admin&password=wrong"
{"detail":"Incorrect username or password"}
PS C:\Users\hebolin> curl.exe -X POST "http://localhost:8000/api/auth/token" -H "X-Forwarded-For: 1.1.1.1" -d "username=admin&password=wrong"
{"detail":"Incorrect username or password"}
PS C:\Users\hebolin> curl.exe -X POST "http://localhost:8000/api/auth/token" -H "X-Forwarded-For: 1.1.1.1" -d "username=admin&password=wrong"
{"detail":"Too many login attempts, please try again later"}
PS C:\Users\hebolin> curl.exe -X POST "http://localhost:8000/api/auth/token" -H "X-Forwarded-For: 2.2.2.2" -d "username=user01&password=Abc123!@"
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxIiwiZXhwIjoxNzY0OTAwNDkxfQ.pvoGaMbj2niKzhYfIra8V_qUyZPGh20OHodgebhgEKI","token_type":"bearer"}
```

Output: Result: WARNING (Rate limit bypassed via X-Forwarded-For header)

Impact: Brute-force protections are weak; attackers may use tools to try millions of passwords without being blocked.

2) **DEF_03: Account lockout policy bypass**

Description: The system's lockout counter resets after any successful login. If an attacker fails to log in to Account A 5 times (reaching the limit), they can simply log in to a different Account B once. This action clears the failure counter, allowing them to immediately resume guessing the password for

Account A.

Test Script:

```
def test_rate_limiting_functionality():
    print("\n--- [Security] Rate Limiting Functionality ---")
    failed_login_attempts.clear()
    setup_user("user1", "Password123!")
    setup_user("user2", "Password123!")
    ip = "1.2.3.4"

    print("Simulating brute force attack from IP...")
    # Fail 5 times with user1
    for i in range(5):
        response = client.post(
            "/api/auth/token",
            data={"username": "user1", "password": "WrongPassword"},
            headers={"X-Forwarded-For": ip}
        )
        if i == 0:
            log_operation("POST", "/api/auth/token", response, data={"username": "user1", "password": "WrongPassword"},
                          headers={"X-Forwarded-For": ip})

    # Try with user2 from same IP
    response = client.post(
        "/api/auth/token",
        data={"username": "user2", "password": "Password123!"},
        headers={"X-Forwarded-For": ip}
    )
```

```
    log_operation("POST", "/api/auth/token", response,
data={"username": "user2", "password": "Password123!"},
                          headers={"X-Forwarded-For": ip})


    if response.status_code == 429:
        print("Result: Passed (IP-based blocking active - User2 blocked
due to shared IP)")
    else:
        print("Result: Failed (User2 not affected by User1 failures)")
```

Output: Result: Failed (User2 not affected by User1 failures)

Impact: The lockout policy is ineffective. Attackers with access to one valid account can indefinitely brute-force other accounts without being permanently locked out.

3) DEF_04：Delete user when user on Dashboard page

Description: When a user is deleted from database, its account should be deactivated, and should be redirect back to login page so user's information will not be exposed.

Test Script:

```
try:
    register(d,"user01","email01@test.com","Abc123!@","Abc123!@")
     login(d,"user01","Abc123!@")
     clear_database_row("user01")
     refresh(d,10)
```

Output: Deleting user will not log the user out, even after several refreshes.

Impact: Terminated or deleted users can still access the system and view sensitive data, creating a significant security breach.

4) DEF_05: Old token works after password change

Description: If I change my password, my old "key" (Token) should stop working. But in this system, the old key still opens the door.

Test Script:

```
def test_token_invalidation_policy():
    print("\n--- [Auth] Token Invalidation Policy ---")
    username = f"user_token_test_{int(time.time())}"
    setup_user(username, "OldPassword123!")

    # 1. Login to get token
    res = client.post("/api/auth/token", data={"username": username, "password": "OldPassword123!"})
    token = res.json()["access_token"]

    # 2. Change password in DB
    db = TestingSessionLocal()
    user = db.query(User).filter(User.username == username).first()
    user.hashed_password = get_password_hash("NewPassword123!")
    db.commit()
    db.close()

    # 3. Try to use the OLD token
    res_check = client.get("/api/auth/users/me", headers={"Authorization": f"Bearer {token}"})
    log_operation("GET", "/api/auth/users/me", res_check, headers={"Authorization": f"Bearer {token}"})

    if res_check.status_code == 200:
        print("Result: INFO (Session remains valid after password change)")
```

```
    else:
        print(f"Result: PASSED (Token invalidated)")
```

Output: Result: REPRODUCED (Old token still valid after password change)

Impact: If a hacker steals a user's token, even changing the password won't guarantee the account's safety.

5) **DEF_06: Login Response Time Consistency**

Description: The system replies very quickly if a username does not exist, but replies slowly if the username exists (because it checks the password). Hackers can guess which usernames are real by measuring response time.

Test Script:

```
def check_login_latency_consistency():
    print("\n--- [Auth] Login Latency Consistency ---")
    setup_user("valid_user", "Password123!")

    # Measure time for non-existent user
    times_non_existent = []
    for _ in range(5):
        start = time.perf_counter()
        client.post("/api/auth/token", data={"username":
"non_existent", "password": "AnyPassword"})
        times_non_existent.append(time.perf_counter() - start)

    avg_non_existent = statistics.mean(times_non_existent)

    # Measure time for valid user with wrong password
    times_valid_user = []
    for _ in range(5):
```

```
            start = time.perf_counter()
            client.post("/api/auth/token", data={"username": "valid_user",
"password": "WrongPassword"})
            times_valid_user.append(time.perf_counter() - start)

    avg_valid_user = statistics.mean(times_valid_user)

    print(f"Avg time (non-existent): {avg_non_existent:.4f}s")
    print(f"Avg time (valid user):      {avg_valid_user:.4f}s")

    if avg_valid_user > avg_non_existent * 2:
        print("Result: WARNING (Significant timing variance detected
- Potential Side Channel)")
    else:
        print("Result: PASSED (Timing consistent)")
```

Output: Result: REPRODUCED (Significant timing difference detected)

Impact: Hackers can easily find out who is registered on the website. And if they get the username, they can use tools to try the password.

4.3 Reliability Defects

1) **DEF_07: Email length boundaries in registration**

   Description: If the user register with a very long email (like 300 letters), the system crashes instead of just saying “Email too long”. This is because the database has a limit for email length (255 chars), however, it is not restricted during user registering. When the code tried to force a long email into the database, the database rejected it, causing the system to crash.

```sql
-- Create users table
CREATE TABLE IF NOT EXISTS users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(100) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    hashed_password VARCHAR(255) NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    is_superuser BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

Test Script:

```
def generate_complex_long_email(total_length=300):
    domain = "@test.com"
    local_part_length = total_length - len(domain)

    if local_part_length < 4:
        raise ValueError("Total length is too short to satisfy complexity requirements.")

    upper_chars = string.ascii_uppercase
    lower_chars = string.ascii_lowercase
    digit_chars = string.digits
    special_chars = "!#$%^&*"
    all_allowed = upper_chars + lower_chars + digit_chars + special_chars

    required_chars = [
        random.choice(upper_chars),
        random.choice(lower_chars),
        random.choice(digit_chars),
        random.choice(special_chars)
    ]
```

```
    remaining_length = local_part_length - len(required_chars)
    random_fill = [random.choice(all_allowed) for _ in
range(remaining_length)]
    local_part_list = required_chars + random_fill
    random.shuffle(local_part_list)
    local_part = "".join(local_part_list)
    full_email = local_part + domain
    return full_email



try:
    enter_register_page(driver)

    long_email = generate_complex_long_email(300)
    input_username_register(driver, "CrashTestUser")
    input_email_register(driver, long_email)
    input_password_register(driver, "Pass123!@")
    input_password_confirm(driver, "Pass123!@")
    click_register_button(driver)
    time.sleep(3)

    page_source = driver.page_source
    if "Internal Server Error" in page_source or "500" in page_source:
        print("\n500 Internal Server Error！ ")
    else "registered successfully" in page_source or "login" in
driver.current_url:
        print("\nRegister success？ ")
```

Output: 500 Internal Server Error!

Stuck in this page forever.

Impact: The website stops working properly when bad data is entered. Attackers could exploit this vulnerability by batch-injecting ultra-long email addresses and cause repeated server crashes.

5. Defect Log

5.1 Section 1: Usability

Defect ID	Test Case ID	Title	Severity	Steps	Expected Results	Actual Results
DEF_01	REG_014	Registration with all inputs invalid	Medium	1. Fill invalid inputs in all fields on Register page. 2. Click Register	All the invalid inputs should be highlighted.	Only email incorrect error is highlighted.

5.2 Section 2: Security

Defect ID	Test case	Title	Severity	Steps	Expected Results	Actual Results
DEF_02	LGN_021	Lockout Mechanism Bypass	High	1. In one IP (1.1.1.1), enter invalid username and password in login page until login is lockout. 2. Switch to another IP (2.2.2.2) within lockout waiting time, and login with username and password.	Account should be locked in both IP, rejecting user login activities.	The user is allowed to keep trying to log in in another IP. System lockout mechanism loses its efficacy.
DEF_03	LGN_022	Account Lockout Policy Bypass	High	1. Enter invalid username and password for account A, and click 'Login' for five times (the limit of system lockout). 2. Login successfully with a different account B. 3. Logout and continue to enter invalid username and	The system should immediately lock the account A after login failure times is more than 5, even after one time	After user successfully logged in account B once, he can continues to try logging in account A until another five times of failure. Lockout counter is reset after each successful login, no matter which account we use.

				password for account A.	of successful login of another account.	
DEF_04	DAT_004	Delete User When User on Dashboard Page.	Medium	1. Login with a valid account and enter dashboard page. 2. Delete user data with database operations. 3. Refresh the page for 10 times.	The system should invalidate the session, log the user out and redirect user to Login page.	After deleting the user account, the system did not log the user out even after multiple times of refreshing. User's information is still exposed.
DEF_05	DAT_005	Password Modification	High	1. Register a new account with password A. 2. Login with password A and automatically copy the success token. 3. Modify password A to password B with database operation. 4. Login again with password A, password B, and try the token.	Password A should success and password B fails. The token should be outdated and not be allowed to use for login.	Both password A and password B are rejected (because password is hashed), but the previous token succeeded and allowed user login.

DEF_06	SEC_006	Login Response Time Consistency	High	1. Enter a valid username and wrong password. 2. Measure response time (Time A) 3. Enter an invalid username. 4. Measure response time (Time B).	Time A and Time B should roughly the same.	Time B is significantly faster than Time A.

5.3 Section 3: Reliability

Defect ID	Test case	Title	Severity	Steps	Expected Results	Actual Results
Def_07	REG_034	Email length boundaries in registration	High	1. Use scripts to generate a 300-char length email that fits email specification. 2. Fill all other fields with valid inputs 3. Click Register	The system should reject the email whose length exceeds the limit	The system did not set any email length constraints, and the register page crashed.