**Lab 09**

**Secure Coding Report**

**SSD-Lab**

**Submitted by:** Abdullah Nadeem

**Roll number:** 22i-1597

**Date:** 30th October 2025

**Table of Contents**

[1. Setup & Libraries 3](#_Toc212755819)

[2. Secure Input Handling (Task 1) 4](#_Toc212755820)

[3. Parameterized Queries (Task 2) 6](#_Toc212755821)

[4. Session Management & CSRF Protection (Task 3) 8](#_Toc212755822)

[5. Secure Error Handling (Task 4) 8](#_Toc212755823)

[6. Secure Password Storage (Task 5) 10](#_Toc212755824)

[7. Application Screenshots 12](#_Toc212755825)

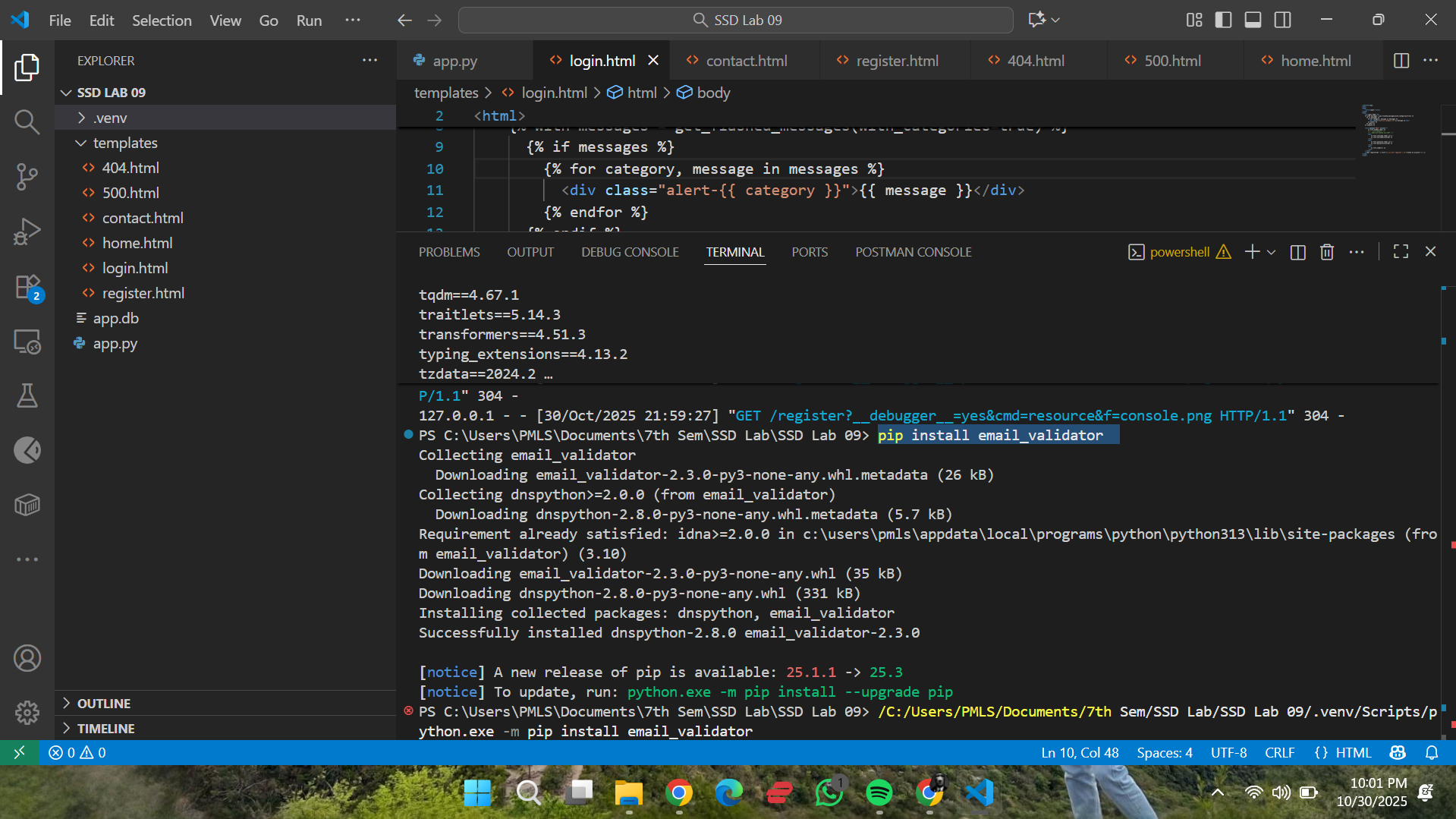
# 1. Setup & Libraries

Task: Install required libraries.

Explanation: Ensured all necessary libraries for the application and security features (Flask, SQLAlchemy, Bcrypt, Flask-WTF, email\_validator) were installed.

A screenshot of a computer

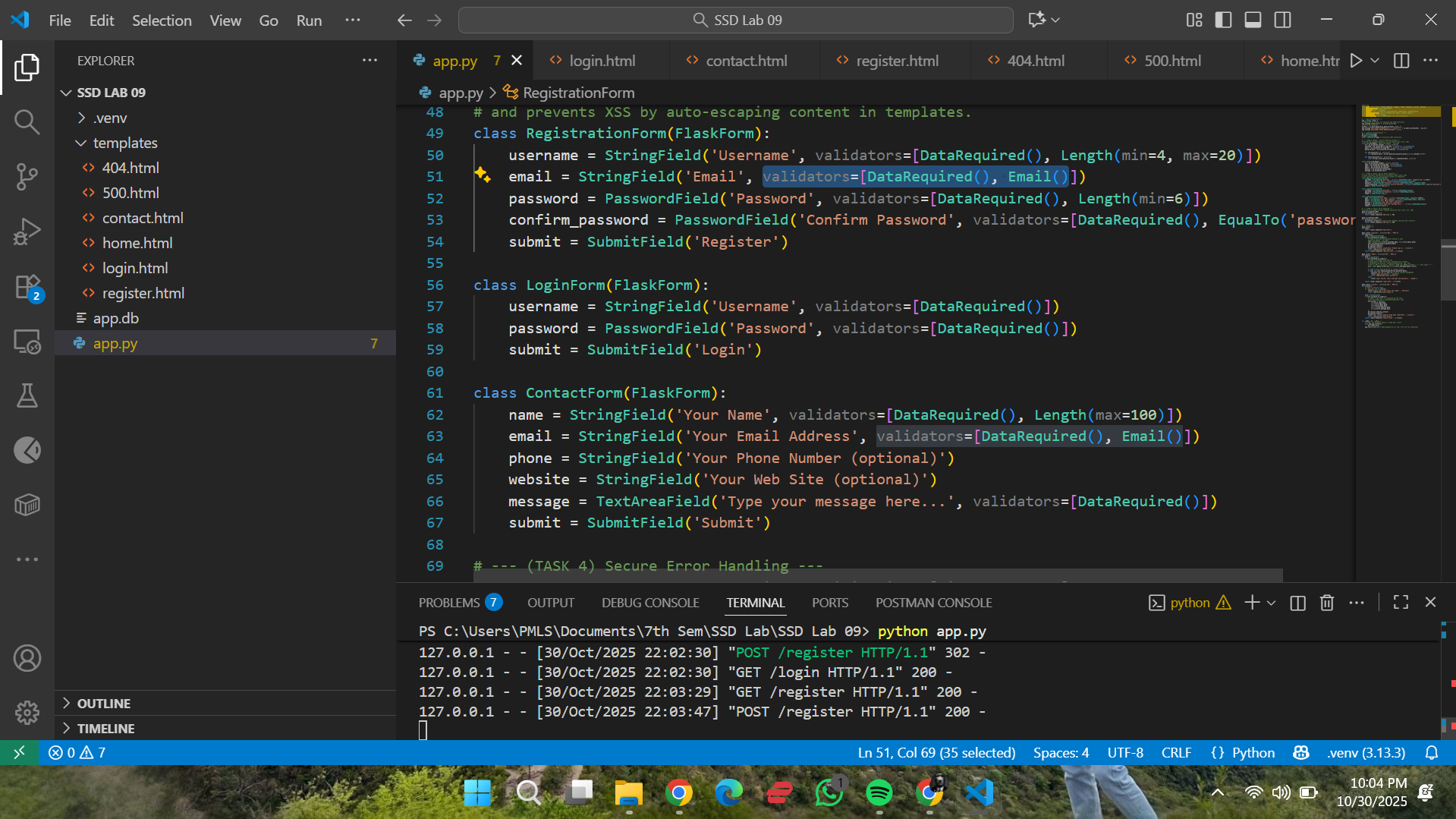
AI-generated content may be incorrect.



# 2. Secure Input Handling (Task 1)

Task: Validate and sanitize all user input.

Explanation: Used Flask-WTF and wtforms.validators to enforce rules (like DataRequired, Email, Length) on all forms, preventing invalid data and XSS.



(Code for 'RegistrationForm' and 'ContactForm' in app.py, highlighting the validators= parts)

A screenshot of a computer

AI-generated content may be incorrect.

(The registration page showing the 'Invalid email address' error after I tried to submit bad data)

# 3. Parameterized Queries (Task 2)

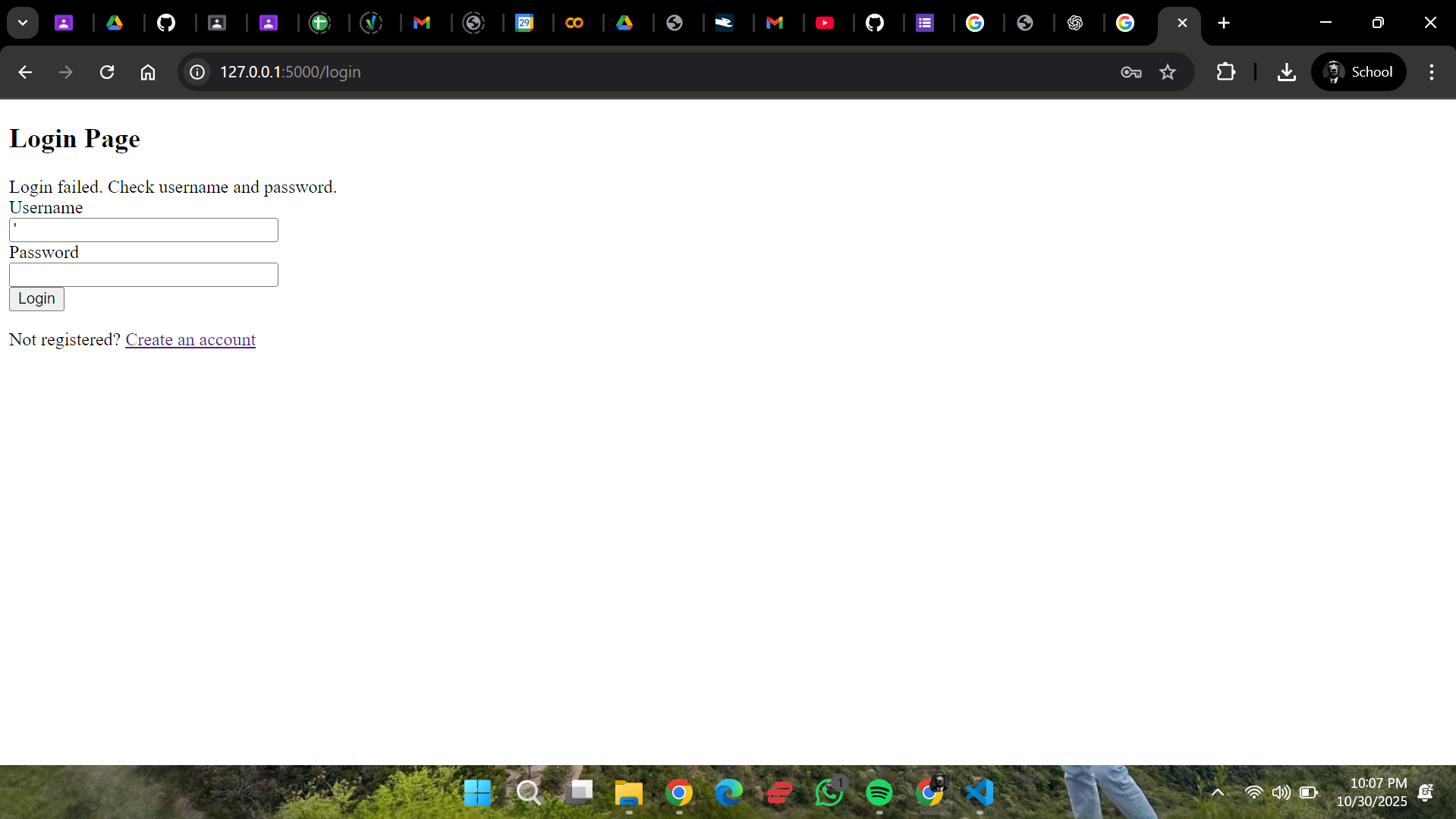
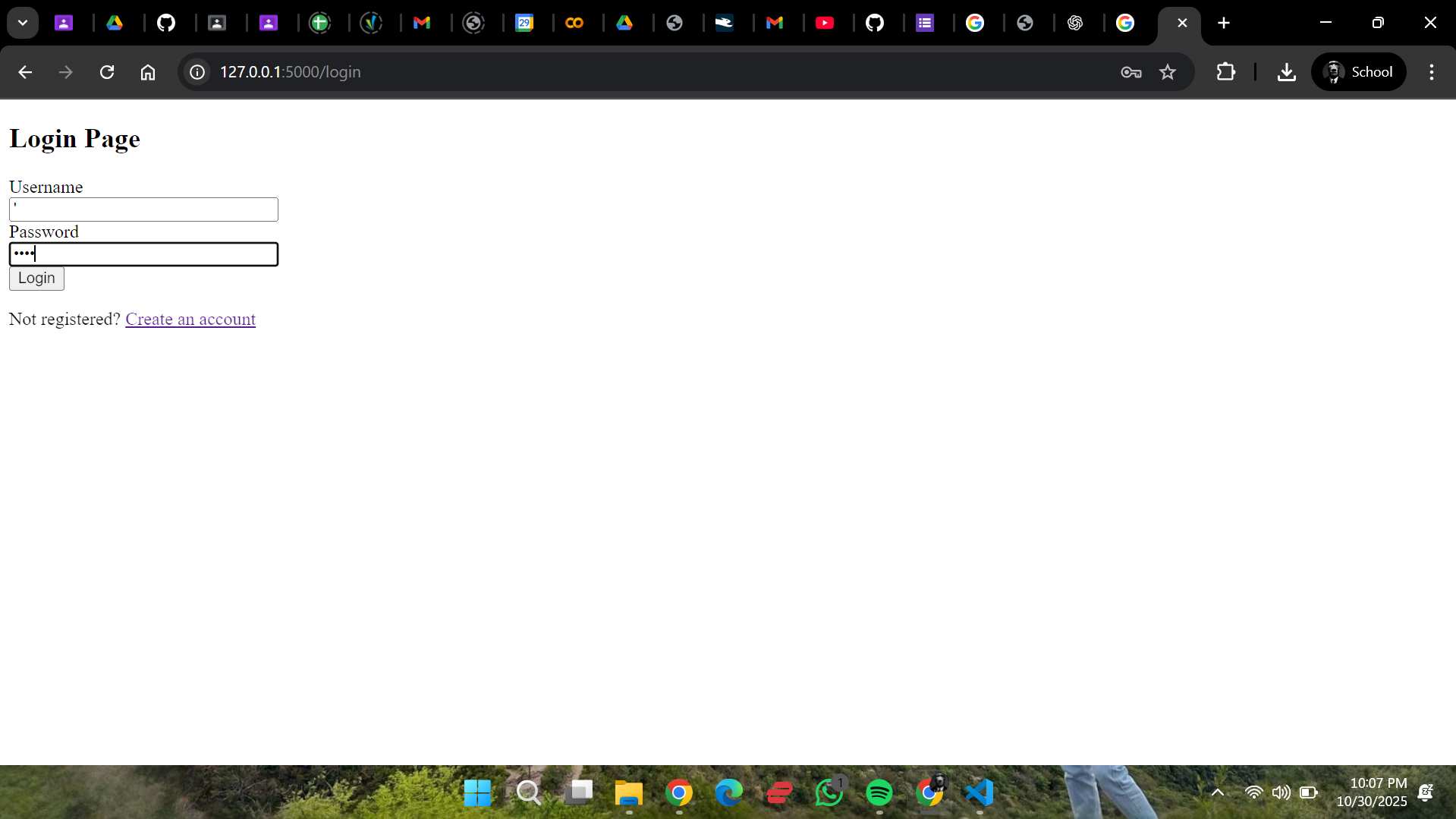
Task: Prevent SQL Injection.

Explanation: Used the SQLAlchemy ORM (.filter\_by()) which automatically creates parameterized queries, treating all user input as text, not as SQL commands.

A screenshot of a computer

AI-generated content may be incorrect.

(Login route code in app.py, highlighting the User.query.filter\_by(...) line)



(The login page showing SQLi attack (' OR '1'='1) failing with the "Login failed" error message)

# 4. Session Management & CSRF Protection (Task 3)

Task: Implement CSRF tokens to protect all forms.

Explanation: Used Flask-WTF to automatically generate a unique, hidden csrf\_token for every form, which is validated on submission to prevent CSRF attacks.

A screenshot of a computer

AI-generated content may be incorrect.

(login.html template code, highlighting the {{ form.hidden\_tag() }} line plus the terminal showing the curl command failing with a "CSRF token missing" or "Bad Request" error)

# 5. Secure Error Handling (Task 4)

Task: Create custom error pages to avoid information disclosure.

Explanation: Implemented custom @errorhandler functions for 404 and 500 errors to show a user-friendly page instead of leaking stack traces or server info.

A screenshot of a computer

AI-generated content may be incorrect.

(@app.errorhandler(404) and @app.errorhandler(500) functions in app.py)

A screenshot of a computer

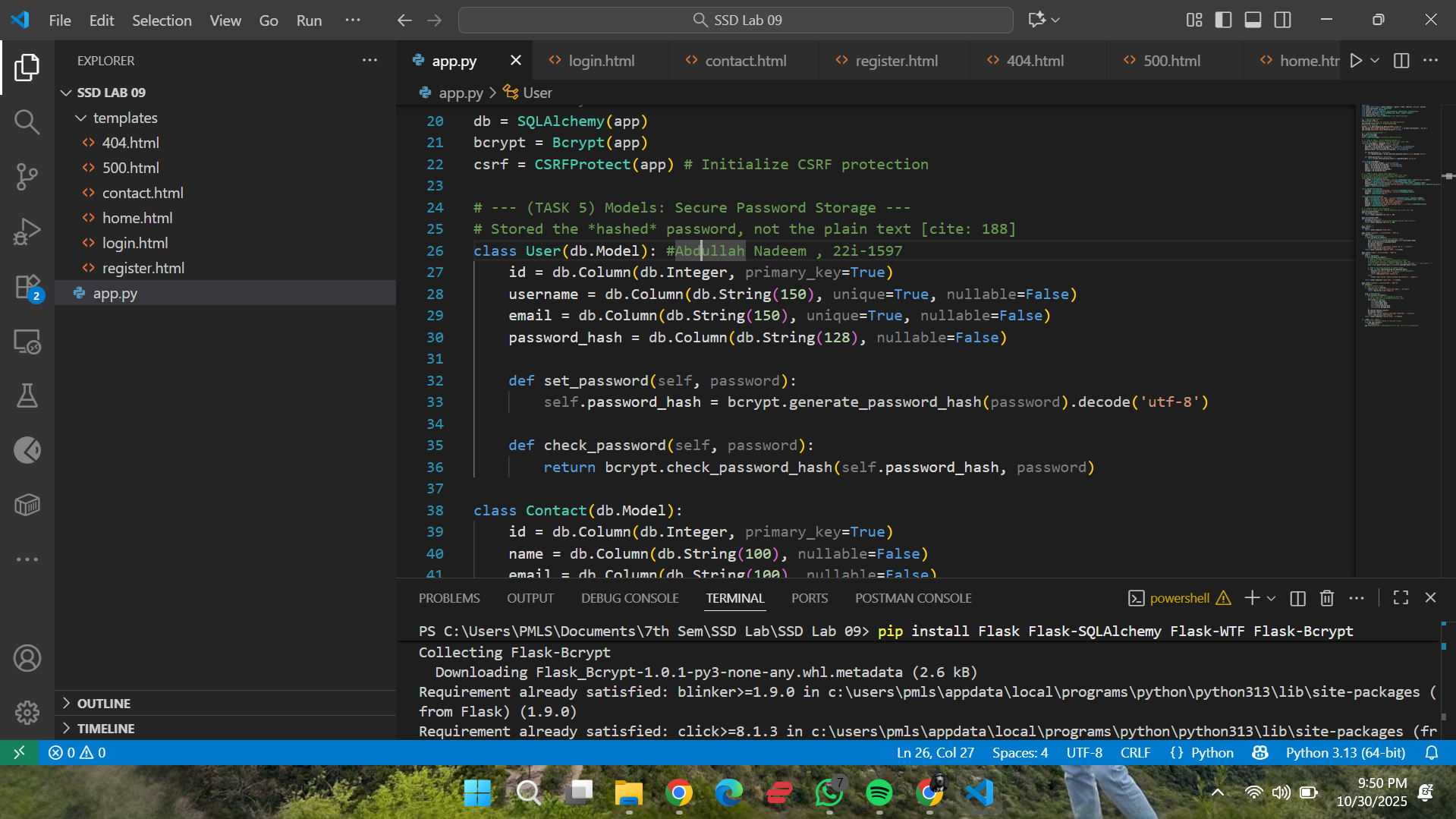
AI-generated content may be incorrect.

(The custom "404 - Page Not Found" page shown in the browser)

# 6. Secure Password Storage (Task 5)

Task: Securely hash and store user passwords.

Explanation: Used Flask-Bcrypt to generate a strong, salted hash of the user's password (set\_password) and safely compare it during login (check\_password).

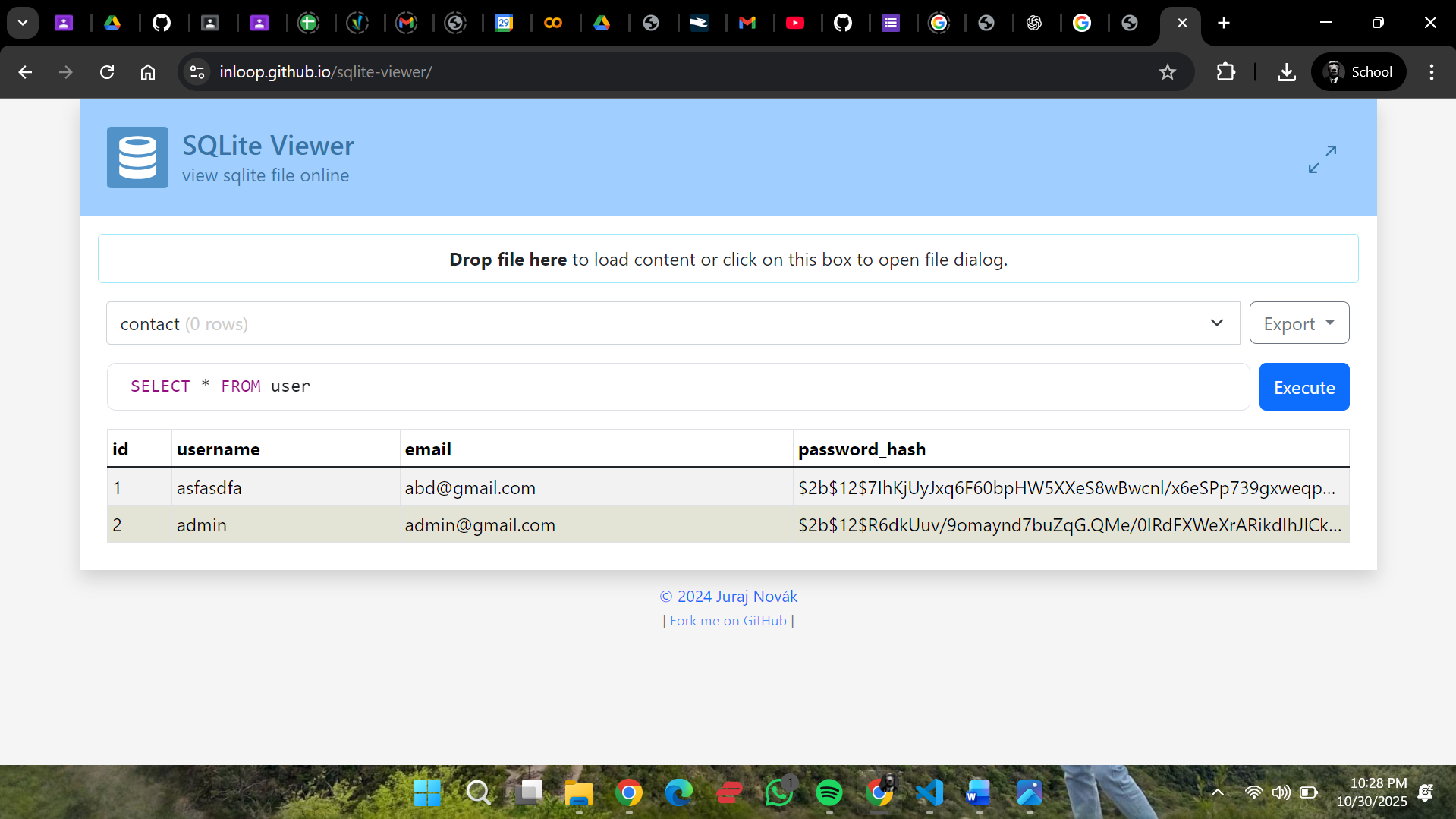


(User model code in app.py, highlighting the password\_hash, set\_password, and check\_password parts)

A computer screen shot of a program

AI-generated content may be incorrect.

(Register route code, highlighting the user.set\_password(form.password.data) line)



(The passwords in the database are encrypted)

# 7. Application Screenshots

Task: Show the functional login page.

Explanation: This is the main login page of the completed Flask application.

A computer screen with a white background

AI-generated content may be incorrect.

(The login.html page rendered in the browser)