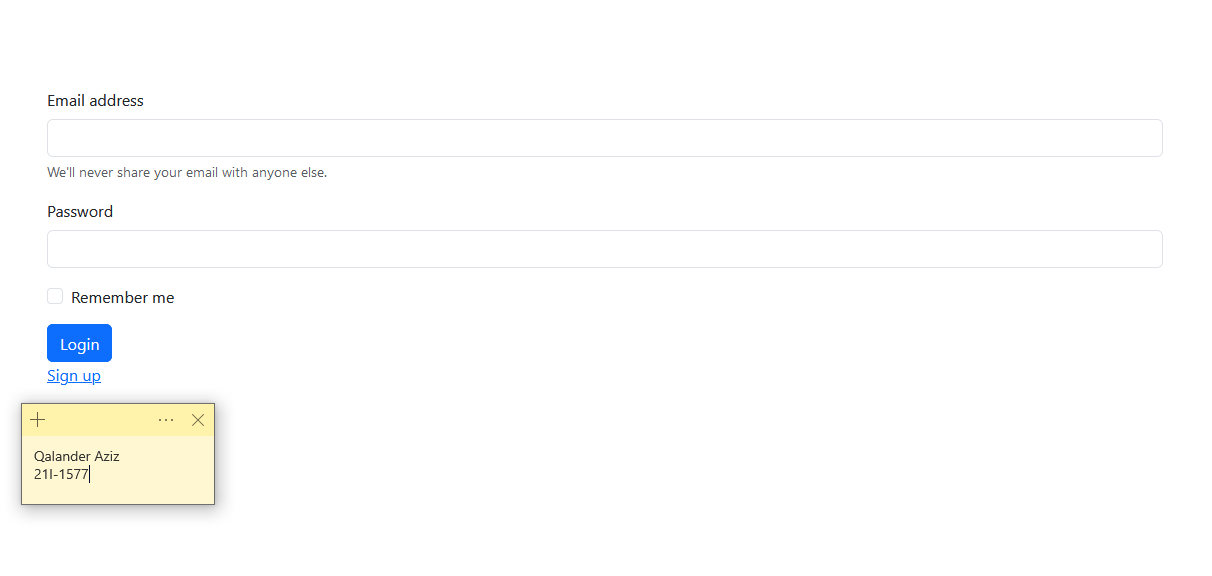
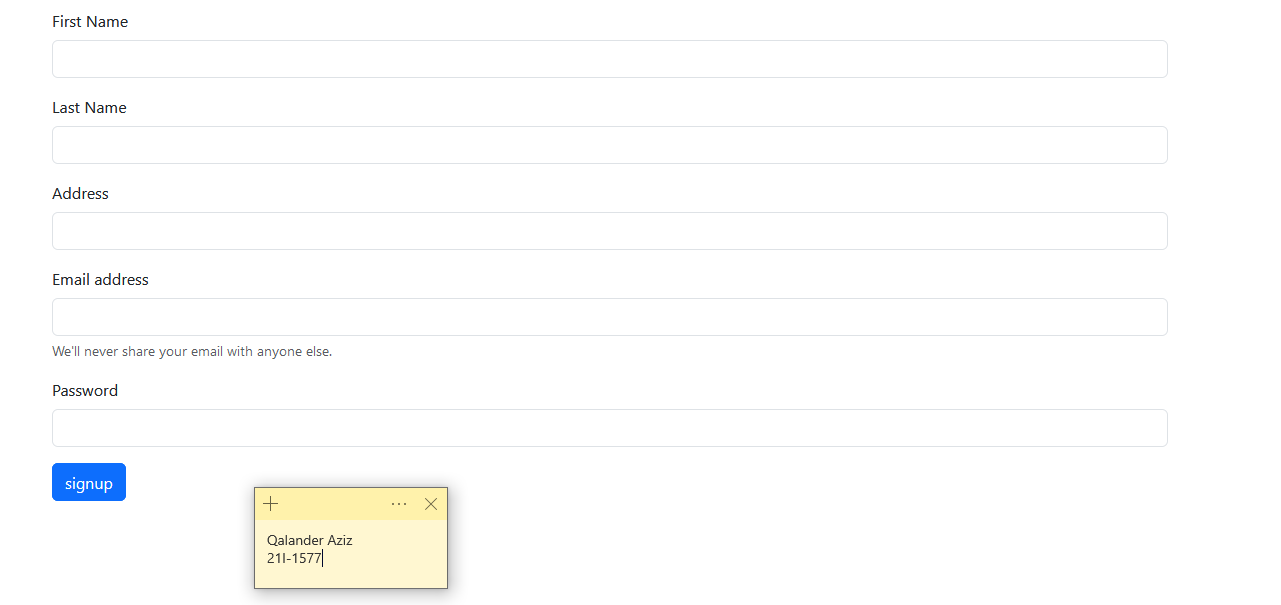
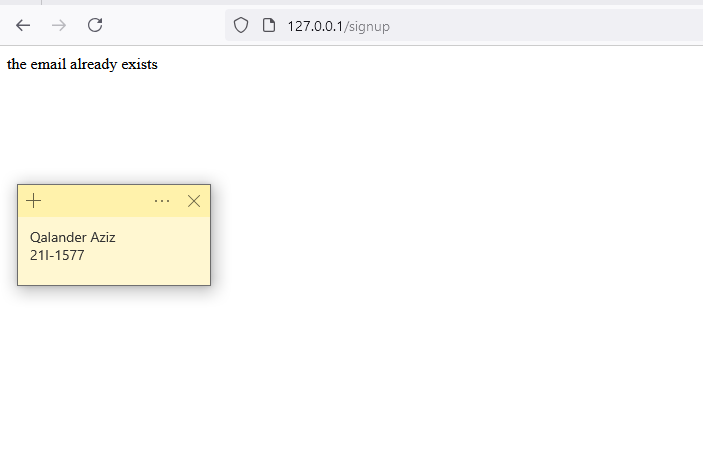
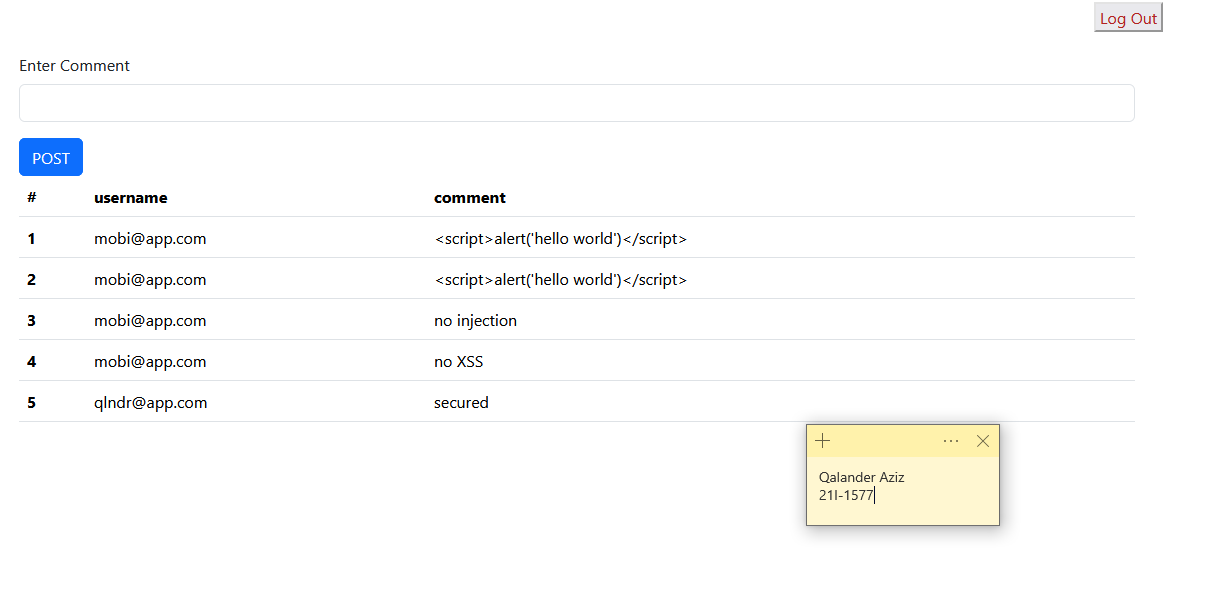
# Application Workflow:

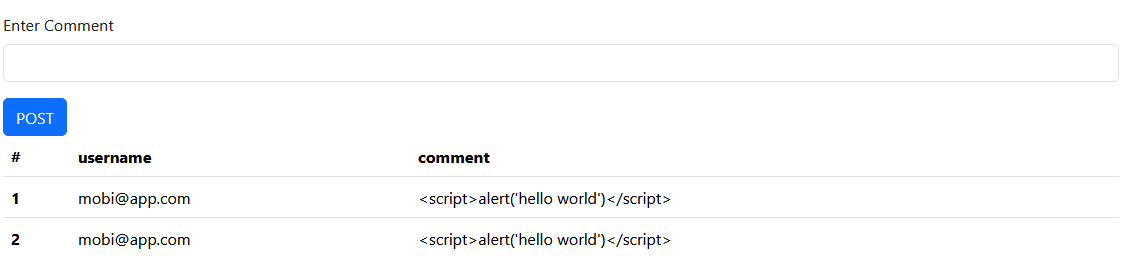
1. Firstly, the user will be redirected to the login page:  
   
2. If the user doesn’t have a account, then the user will sign up first and redirected to login page to enter credentials:  
   
3. If the entered email already exists in the database, then it will display the following error page:  
   
4. If the user entered the valid credentials on the login page, then the user will be redirected to the page, where he can comment:  
   
5. If the user clicks on Log Out button, the session related to the user terminated and, he/she will be redirected to the login page.

# Implemented Security Features: 1. Secure Input Handling:

**Feature:** Input from users is captured using request.form for login, signup, and comments.

**Technique:** Input validation is applied by verifying the presence of required fields (e.g., username and password) and sanitization of the inputs.

**Security Improvement:** This prevents issues like empty input, ensuring that only valid, non-empty data is processed. Additionally, future improvements could include limiting input length and using whitelisting for special characters to guard against injection attacks.



No script executed when the page reloads.

## 2. Use of Parameterized Queries:

**Feature:** The database queries are generated using SQLAlchemy’s ORM methods, such as filter\_by() and add().

**Technique:** SQLAlchemy automatically escapes user inputs, avoiding the risk of SQL injection.

**Security Improvement:** By relying on SQLAlchemy’s ORM, the code avoids manually concatenating SQL queries, which is a common vulnerability to SQL injection. This ensures that the queries are always safe, even when dealing with user-provided data.

## 3. Session Management:

**Feature:** User sessions are handled using Flask’s session object, with session expiration set to 30 minutes.

**Technique:** A secret\_key is defined, and Flask uses it to sign session cookies. The session is cleared during logout (session.pop()).

**Security Improvement:** The secret key ensures that the session data is cryptographically secure, and session expiration mitigates the risk of prolonged exposure if a session is hijacked. Additionally, by clearing the session on logout, the application reduces the risk of session fixation attacks.

## 4. Secure Error Handling:

**Feature:** Error handling for issues like existing users (signup) is addressed by returning a simple error message.

**Technique:** Custom error messages (e.g., "the email already exists") are used without exposing stack traces or internal application details.

**Security Improvement:** This technique prevents leaking sensitive information about the internal structure of the app, which could be useful to attackers for further exploitation. By providing user-friendly error messages, the app remains secure without revealing technical details.

## 5. Secure Password Storage:

**Feature:** Passwords are hashed using bcrypt and stored in the database.

**Technique:** The bcrypt algorithm is used for hashing passwords with a random salt (bcrypt.gensalt()), making it computationally expensive to crack.

**Security Improvement:** Storing hashed passwords (instead of plain text) prevents attackers from easily accessing users' passwords in case of a database breach. The bcrypt algorithm is also resistant to brute-force attacks due to its computational complexity.