SOHAIB KHAN
2022551
CYBER SECURITY
SSD WEEK : 3

Week 3: System Architecture & Secure Design

In this week, I will need to focus on creating the system architecture for my application and design the security measures necessary for its functionality. Here's an overview of what I have completed for **System Architecture & Secure Design**:

---

1. System Architecture Diagrams

Objective: Create a clear visual representation of the structure of your application, showing how various components (database, server, client, etc.) interact.

Steps:

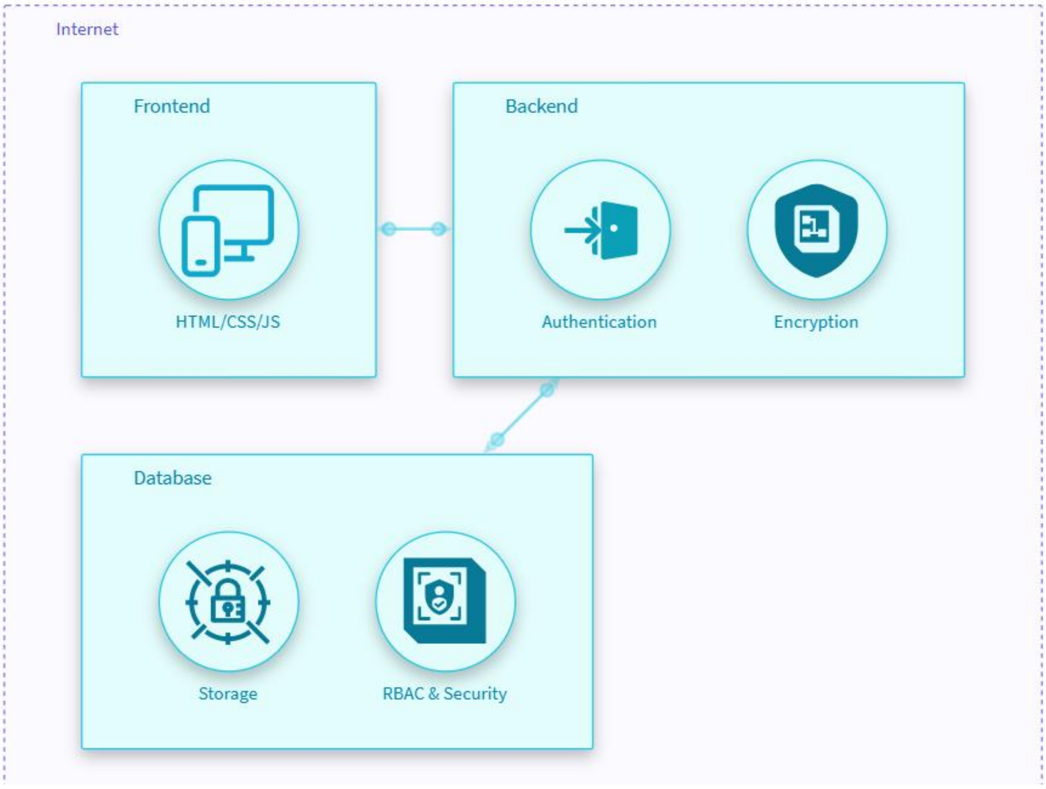
Backend (Flask): Define the flow of data, including the database interactions, user authentication process, and note management (CRUD operations).

Frontend (HTML/CSS): Illustrate how the UI connects to the backend (via routes for login, dashboard, etc.).

Security Architecture: Show how security layers are implemented, like user authentication, password hashing, encryption of notes, and CSRF protection.

Tools for Architecture Diagrams

Iruisrisk



2. Security Controls

Authentication:

Implement **Flask-Login** for user session management, which ensures that users are logged in before accessing protected routes (like the dashboard).

Flask-WTF (with CSRF protection) to safeguard your forms from CSRF attacks.

Flask-Bcrypt to securely hash and verify passwords.

Encryption:

Fernet Encryption for encrypting and decrypting notes stored in the database.

This encryption ensures that even if the database is compromised, the note contents remain unreadable without the encryption key.

Access Control:

Only authenticated users should be able to access their dashboard and view/edit/delete their notes.

Flask-Login's @login_required decorator ensures that users must be logged in to access the dashboard.

Notes should be linked to a specific user, ensuring that users can only access their own data.

---

3. Security Design Measures

Password Storage:

Use **bcrypt** to hash passwords before storing them in the database. This prevents storing passwords in plain text and makes the application more secure.

Implement measures to prevent **brute-force attacks**, such as adding rate-limiting for login attempts (like you did with Flask-Limiter).

HTTPS:

In a production environment, configure HTTPS to ensure secure communication between the client and server. This can be done with a production server like **Gunicorn** behind **Nginx** or **Apache**.

Data Encryption:

As mentioned, encrypt the sensitive data (e.g., notes) using Fernet encryption. This ensures that even if an attacker gains access to the database, the note content remains encrypted.

Secure Headers & CSRF Protection:

Use secure HTTP headers (e.g., Strict-Transport-Security) and ensure CSRF protection is enabled on forms, as you have done using Flask-WTF.
