**National University of Computer & Emerging Sciences, Islamabad**

****

**Spring - 2025**

**Information Security Course Project**

**Submitted By:**

Maaheen Sidiqui 21i-0278

Maryam Amjad 21i-0309

Fatima Tahir 21i-2701

**Submitted To:**

Ma’am Noshina Tariq

## **Git Repository Link:**

Link: <https://github.com/Maryam7892/Information_Security_Project.git>

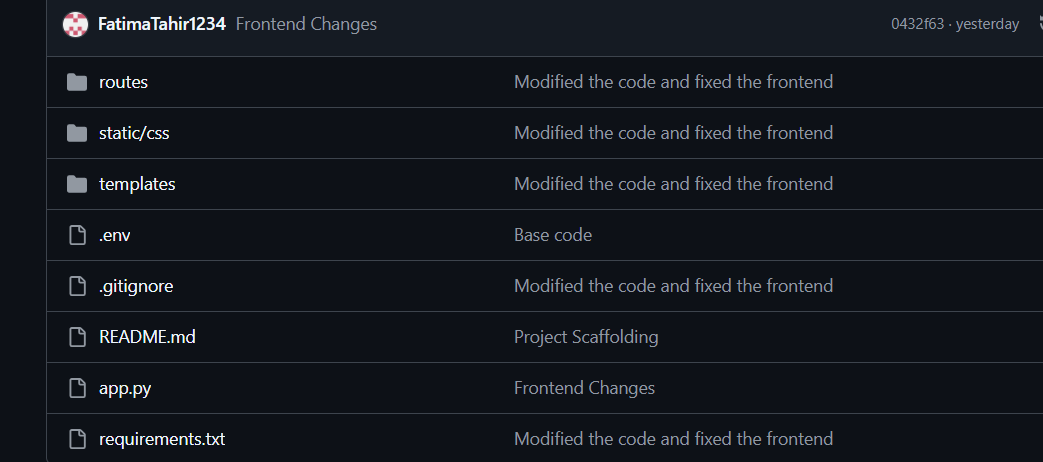
## **Overview:**

The goal of this project is to develop a secure web application using Flask that demonstrates Post-Quantum Cryptography (PQC) techniques for encrypting and decrypting data. Students will integrate a PQC algorithm (e.g., Kyber, Dilithium, or any NIST finalist) and implement a user interface to showcase key generation, encryption, and decryption functionality.

### **Tools and Technologies:**

* Programming Language: Python 3.x
* Framework: Flask
* Cryptography: Use pqcrypto or similar libraries
* Frontend: HTML, CSS, JavaScript (Bootstrap optional)
* Security: Flask-Talisman, Flask-WTF (optional)
* Version Control: Git + GitHub

### **Project Scaffolding:**



## **Methodology:**

To develop the Post-Quantum Cryptography (PQC) web application, we began by setting up a Python virtual environment and installing the necessary dependencies, including Flask for the web framework, Flask-Talisman for secure HTTP headers, and Python-dotenv for managing environment variables. The application was organized with a clean folder structure, separating templates, static files, and routes to follow best coding practices.

**Key Generation:**

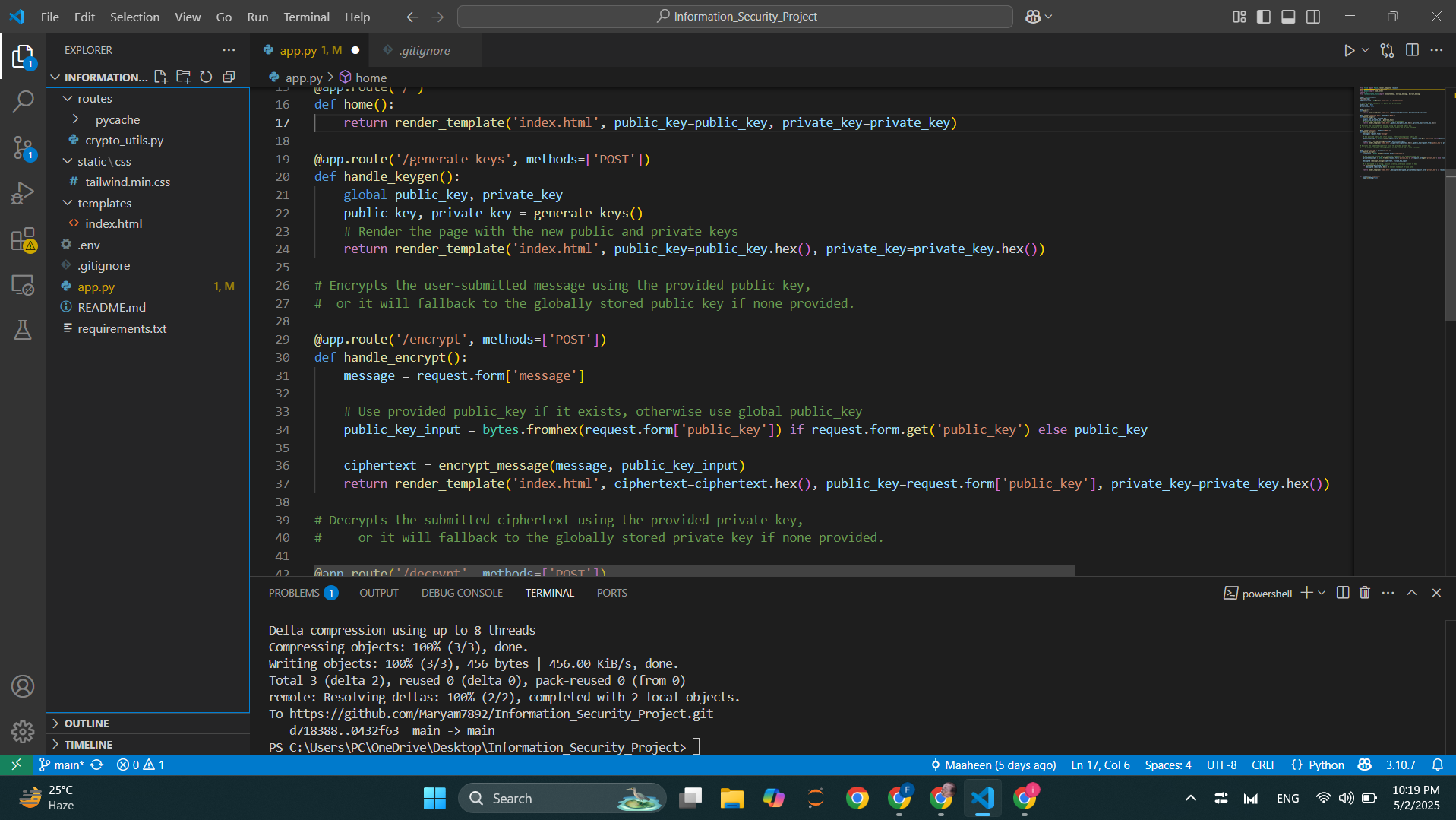
For cryptographic operations, we created a placeholder module named crypto utlis containing functions for generating public-private key pairs, encrypting messages, and decrypting ciphertext. In the main Flask app, global variables temporarily stored the keys for each session. The route renders the homepage, while the generate keys function is to create and display a new key pair in hexadecimal format.

**Encryption/Decryption:**

The /encrypt and decrypt route handled message encryption and decryption respectively. The encryption route accepted a message and public key, using the stored key if none was provided, then returned the ciphertext in hexadecimal form. Similarly, the decryption route accepted a ciphertext and private key, with the option to use the stored key if needed, and displayed the decrypted message on the homepage.

**FrontEnd:**

On the front end, an HTML template created a simple, clean interface for key generation, encryption, and decryption, styled with basic CSS. To enhance security, Flask-Talisman was integrated, enforcing secure HTTP headers and protecting against common web threats. This methodology successfully demonstrated key PQC concepts in a secure and interactive web application.



## **Static Demo:**

# add pic

ThankYou!