

Verveox Technologies

Python Internship - Week Three Task

Task Title: Signup Form with SQLite Integration

SUBMITTED BY: Gopika G

SUBMITTED TO: Adith Sir

INTERN ID: VERVEOX25PY32

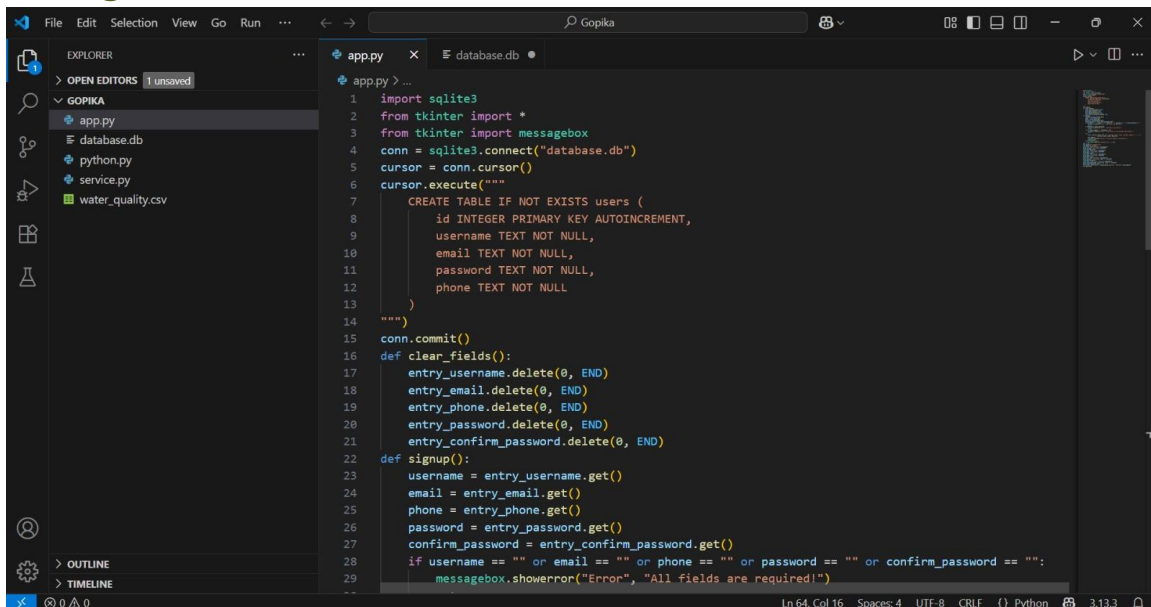
SUBMITTED ON: 26 / 06 / 2025

1. Objective

The primary objective of this task is to design and implement a user registration system using Python's Flask web framework, integrating a responsive HTML form with a secure backend powered by SQLite. This task emphasizes full-stack development principles by guiding the creation of a dynamic web form that validates user input—specifically password confirmation—and persistently stores user data in a structured relational database. It aims to strengthen foundational skills in web application development, data handling, and backend integration, while reinforcing best practices in form validation, user experience, and secure data storage.

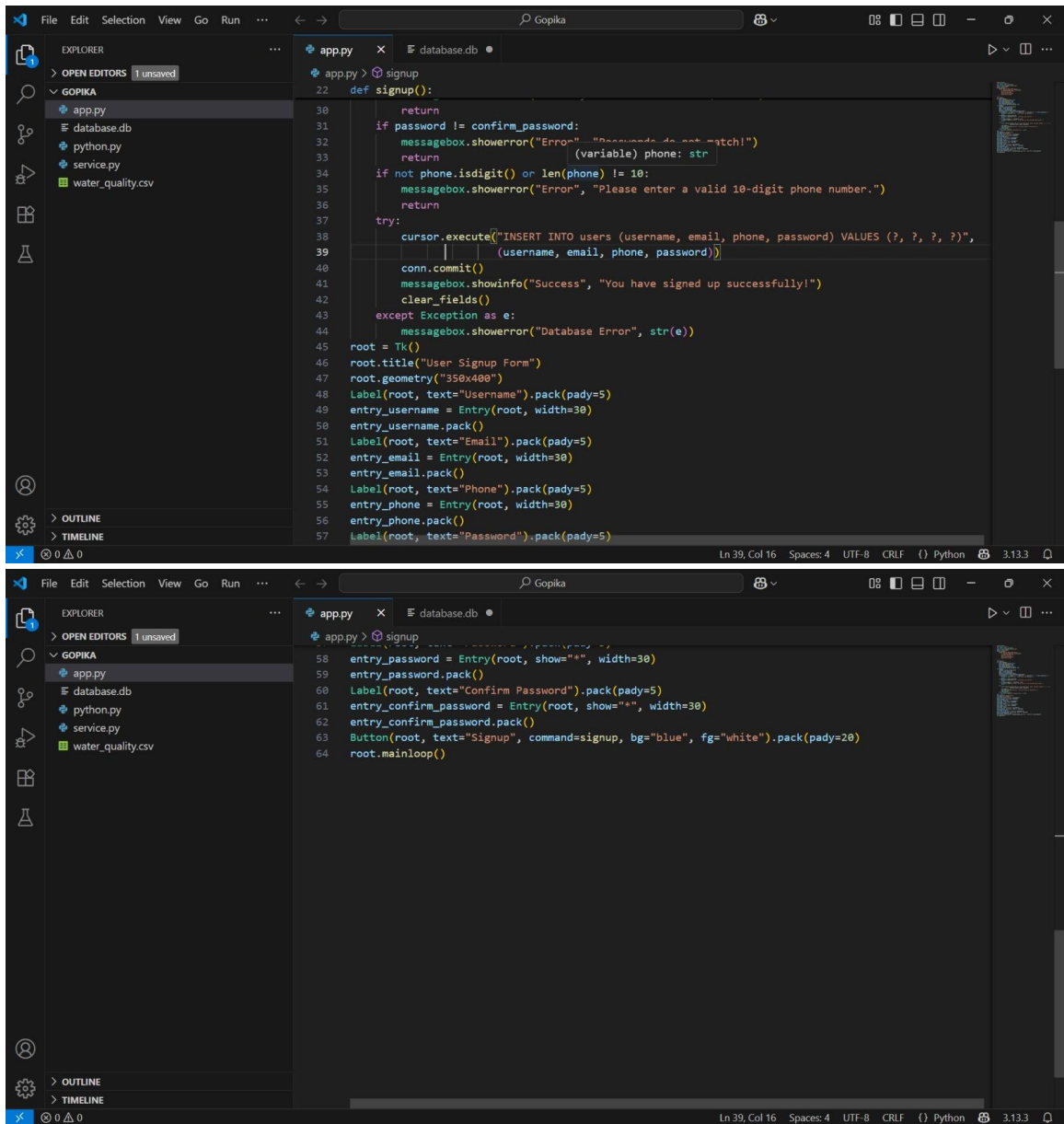
In addition, the task fosters a deeper understanding of how client-server communication operates in a real-world web application scenario. By applying modular programming practices, implementing conditional logic for error handling, and maintaining data consistency through SQL operations, this project cultivates both technical proficiency and an engineering mindset—critical competencies for building scalable and maintainable applications. The exercise also serves as a stepping stone toward more advanced topics such as user authentication, encryption, and session management.

2. Program code

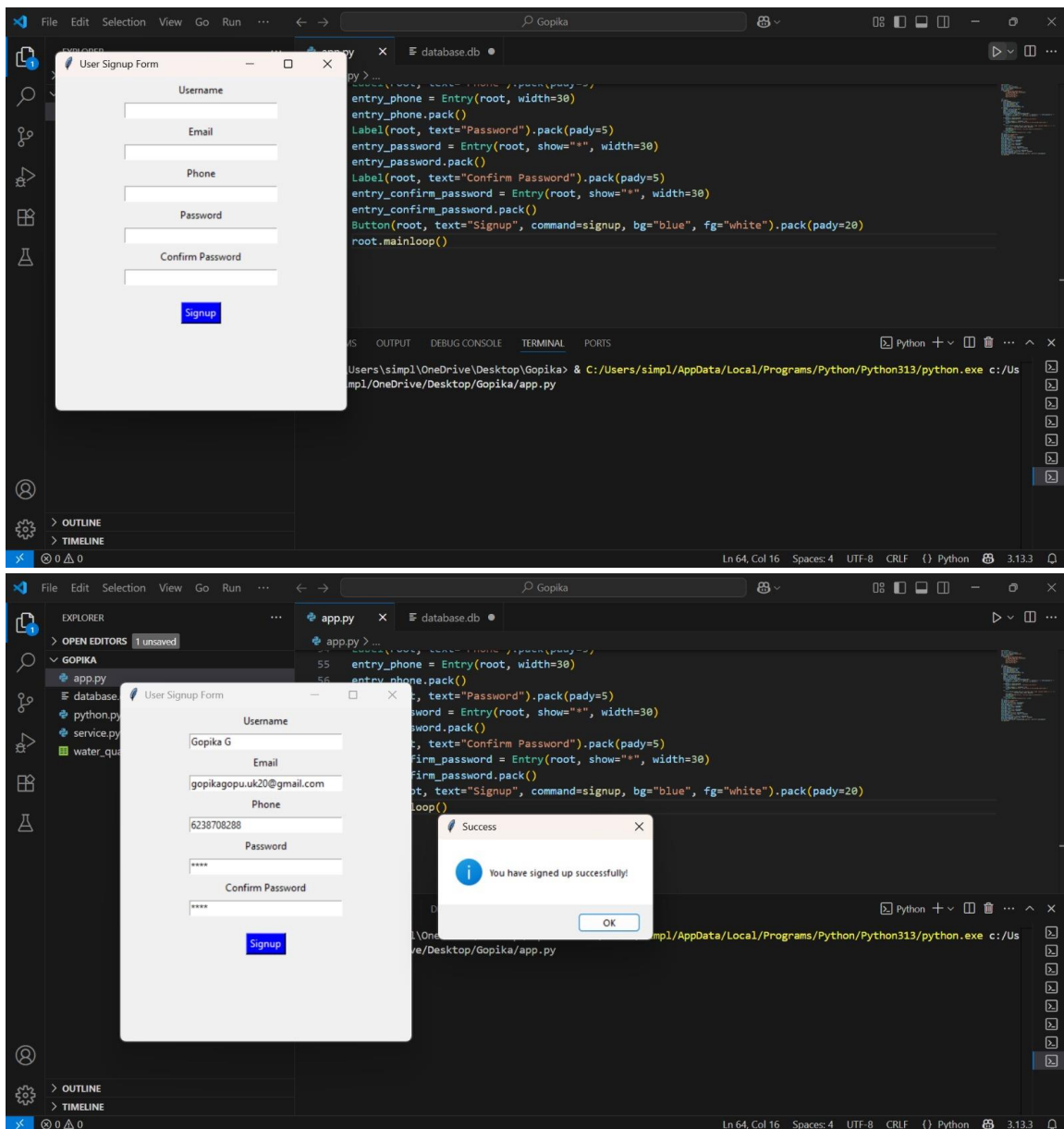
A screenshot of a code editor window titled 'Gopika'. The editor shows a Python file named 'app.py' with the following code:

```
1 import sqlite3
2 from tkinter import *
3 from tkinter import messagebox
4 conn = sqlite3.connect("database.db")
5 cursor = conn.cursor()
6 cursor.execute("""
7     CREATE TABLE IF NOT EXISTS users (
8         id INTEGER PRIMARY KEY AUTOINCREMENT,
9         username TEXT NOT NULL,
10        email TEXT NOT NULL,
11        password TEXT NOT NULL,
12        phone TEXT NOT NULL
13    )
14 """)
15 conn.commit()
16 def clear_fields():
17     entry_username.delete(0, END)
18     entry_email.delete(0, END)
19     entry_phone.delete(0, END)
20     entry_password.delete(0, END)
21     entry_confirm_password.delete(0, END)
22 def signup():
23     username = entry_username.get()
24     email = entry_email.get()
25     phone = entry_phone.get()
26     password = entry_password.get()
27     confirm_password = entry_confirm_password.get()
28     if username == "" or email == "" or phone == "" or password == "" or confirm_password == "":
29         messagebox.showerror("Error", "All fields are required!")
```

The code editor interface includes a sidebar with a file explorer showing 'GOPIKA' and files like 'app.py', 'database.db', 'python.py', 'service.py', and 'water_quality.csv'. The bottom status bar indicates 'Ln 64, Col 16', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', and '3.13.3'.



3. Program Output



4. Code Explanation

This project is a simple web application that allows users to sign up by filling out a form. It uses Flask for the backend, HTML for the frontend, and SQLite to store the data. The code is organized into three main parts: the backend logic, the signup form, and the confirmation page.

❖ Backend Logic (app.py)

The core of the application is built with Flask. The app is set up with a secret key so we can display flash messages (like password errors) to users. A helper function called `init_db()` is used to create the SQLite database (`users.db`) and a `users` table if they don't already exist.

The main route (`/`) handles two things:

- It displays the signup form (when the user first visits the page).
- It processes the form data when submitted (POST request).

When the form is submitted, the app checks that both passwords match. If they don't, an error message appears. If they do match, the user's details are saved into the SQLite database and the app redirects to the success page.

❖ The Signup Form (`signup.html`)

This HTML page contains a simple, user-friendly form with fields for username, email, phone number, password, and confirm password. All fields are required. If there's a problem—like mismatched passwords—an error message appears above the form using Flask's flash messaging system. Basic CSS is used to keep the layout clean and centered.

❖ The Success Page (`success.html`)

Once the user submits valid data, they're redirected to this page. It confirms that their registration was successful with a simple message. It's intentionally minimal to keep the user experience straightforward.

❖ SQLite Database

We use SQLite as a lightweight database to store user data. It's easy to set up and doesn't need a separate server. Data is stored securely using a parameterized SQL query to prevent SQL injection. The connection to the database is managed using Python's `with` statement to ensure it closes automatically.

❖ Validation & Error Handling

Before saving anything, the app checks if the two password fields match. If they don't, it flashes an error and stops the submission. This helps avoid incorrect or incomplete data. Flash messages keep the user informed without interrupting their flow.

❖ Navigation & Redirects

After successful registration, the user is redirected to the `/success` route using Flask's `redirect()` and `url_for()` functions. This keeps the app structured and separates the form and confirmation steps clearly.

5. Challenges Faced

- **Password Mismatch Validation**

Initially, the form accepted mismatched passwords. This was resolved by adding a server-side check and displaying an error using Flask's flash messaging.

- **Database Table Creation**

The users table didn't generate on first run. The issue was fixed by adding an `init_db()` function to create the table when the app starts.

- **Flash Messages Not Displaying**

Error messages weren't showing due to a missing secret key and incorrect HTML syntax. Adding `app.secret_key` and proper Jinja blocks resolved it.

- **Redirection Failure**

After form submission, users weren't redirected to the success page. Adding `return redirect(url_for('success'))` fixed the issue.

- **Poor Form Layout**

The form appeared cluttered. Basic CSS styling was applied to improve structure and user experience.

6. Conclusion

Developing this signup system marked a meaningful step in bridging the gap between theory and practical web development. From designing a user-friendly interface to implementing backend validation and database integration, the project offered a comprehensive view of how individual components in a web application work together to deliver a seamless experience.

The challenges faced along the way—ranging from subtle logic errors to UI refinements—were not setbacks but learning moments that deepened my problem-solving skills and sharpened my attention to detail. More importantly, this task reinforced the value of thoughtful design: building applications that not only work, but are also reliable, user-focused, and maintainable.

By the end of this task, I've gained confidence in using Flask to build dynamic web applications, and I now have a clearer understanding of what it takes to move from an idea to a fully functional, data-driven product.