PROJECT REPORT

Python User Registration App

Submitted by: Utkarsh…Registration Number = 25BAI11231

Submitted to: Prof. MK Jayanthi Kannan

Date: 24 November 2025

Cover Page

Project Title: Python User Registration App

Project Goal: Design and implement a Python application to securely collect, validate, and persist user registration data.

Author: Utkarsh

Date of Submission: 24 November 2025

Project Introduction

A registration form is a fundamental component of most applications—web, desktop, and mobile. It allows users to enter personal information which is then validated and securely stored for future login or identification.

This document outlines the design and implementation of an "User Registration Form" dedicated to the Administrative forms intake, such as Government Jobs, Institutional application for admissions, applying for an interview, online/offline competitions, Lucky Draw forms, creating a digital account in banks or on any digital services like Gmail etc.

All these are essential part of our life and these often requires the new registration related works and removing old, unrequired data and people.

The application provides a simple graphical user interface (GUI) where users can input required personal details

Problem Statement

Many beginner-level Python learners struggle to understand GUI development and how data can be collected and stored persistently.

The primary problem this project addresses is the need for a **digital, standardized, and secure pipeline** to acquire, validate, and store user registration records. This ensures that the collected data is correct, consistent, and immediately accessible for future use.

Thus, there is a need for managing these information and data to be simple, efficient, enhanced and secured.

Functional Requirements

The system must perform the following actions:

- The system must display a user registration form with fields for Full Name, Email Address and Password.
- The system must validate the Email Address format
- The system must ensure all required fields (Name, Email, Password) are filled before submission.
- Upon successful validation and submission, the system must securely store the user's data (excluding the raw password) in a persistent database.
- The system must display a clear "Registration Successful" message upon successful data storage.
- The system must display specific error messages for any validation failure (e.g., "Invalid Email Format").

Non-Functional Requirements

- The GUI must be intuitive, responsive, and easy to navigate for non-technical users.
- Passwords must be stored using a cryptographic hash function like bcrypt, rather than plain text.
- The application should be executable on all major operating systems (Windows, macOS, Linux) where Python and its dependencies are installed.

System Architecture

The system follows a simple layered architecture:

- User Interface Layer : kivy (to collect user details)
- Validation Layer (functions to check correctness of inputs)
- Data Storage Layer (store user data)
- Notification Layer (Message Boxes)

Architecture Flow

User Input → Validation → Storage → Confirmation/Errors.

Design Diagrams

- **Use Case:** Fill Form → Validate Data → Store User Data → Show Confirmation.
- **Workflow diagram:** User → Registration Form → Validation Module → Storage System → Confirmation Message.
- **Sequence Flowchart:**
  Start
  Display registration form
  User enters data
  Validate inputs
  If invalid → Show error → Go back
  If valid → Save to file
  Show success message
  End

Design Decisions and Rationale

Decision	**Rationale**
Programming Language: Python	Chosen for its extensive library support, and excellent readability,

	making the project maintainable and scalable.
GUI Framework: kivy	Chosen as it is the standard, lightweight, and pre-installed GUI library for Python, minimizing external dependencies.
Database: SQLite	Chosen for its serverless, file-based architecture. It is ideal for small to medium-scale desktop applications, requiring zero configuration and offering transactional database integrity.

Implementation Details:

- **Tools and Technologies used:** Python 3.13 64-bit, Tkinter for GUI,
  File system for storage, SQLite.
- **`validator.py`**: Contains all logic functions, including email format matching, password checks and checking for empty fields.
- **Major Code Modules:** GUI Module, Validation Module, Storage
  Module, Message Module.

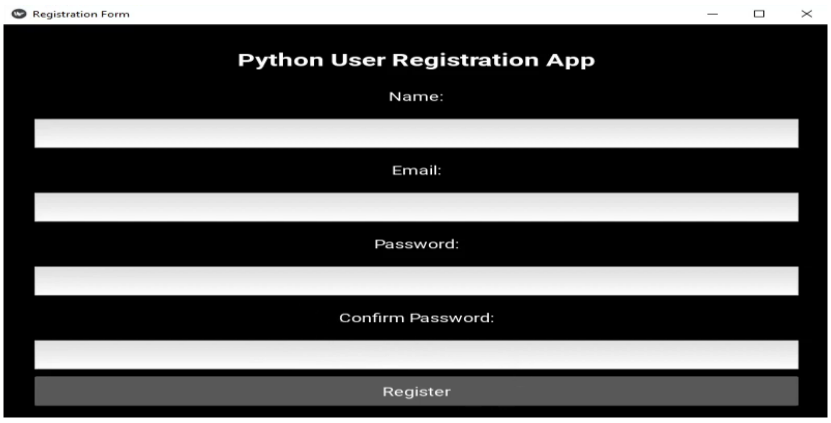
Program Execution and Results:

- The project successfully meets all functional requirements.
- A user is able to interact with the registration form and all submitted data is validated in real-time.

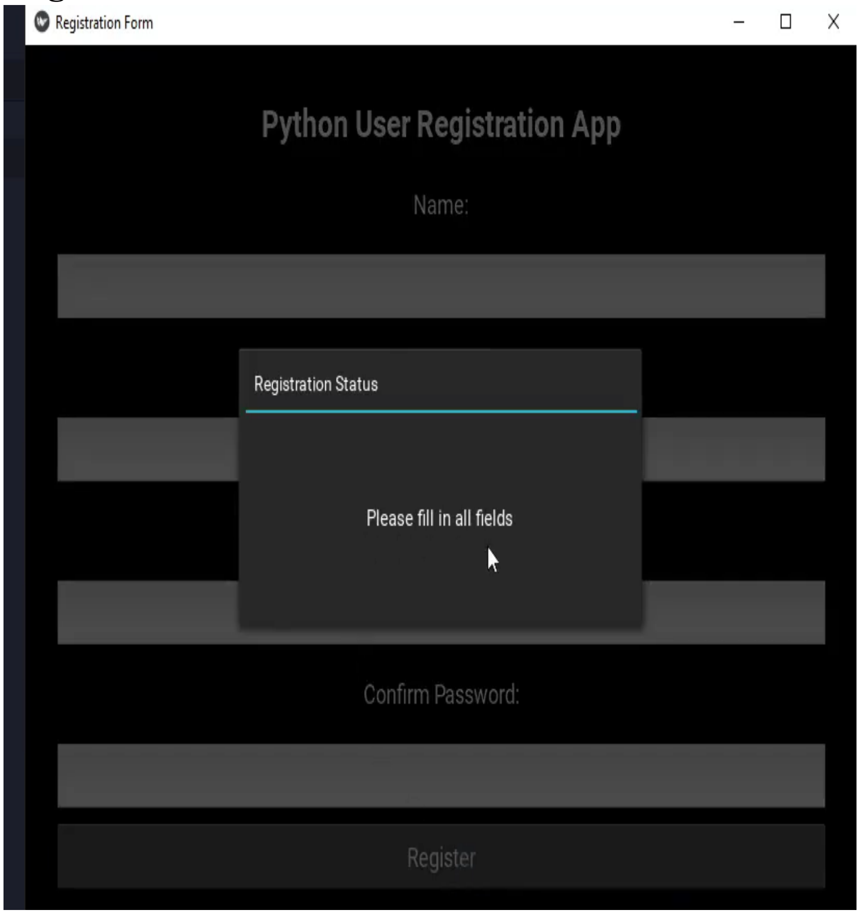
  Upon successful submission, a new record, including the securely hashed password, is correctly inserted into the users.db SQLite database.

Here, I am pasting my all the results of my programs when executed:

1. **Development of Python User Registration App**

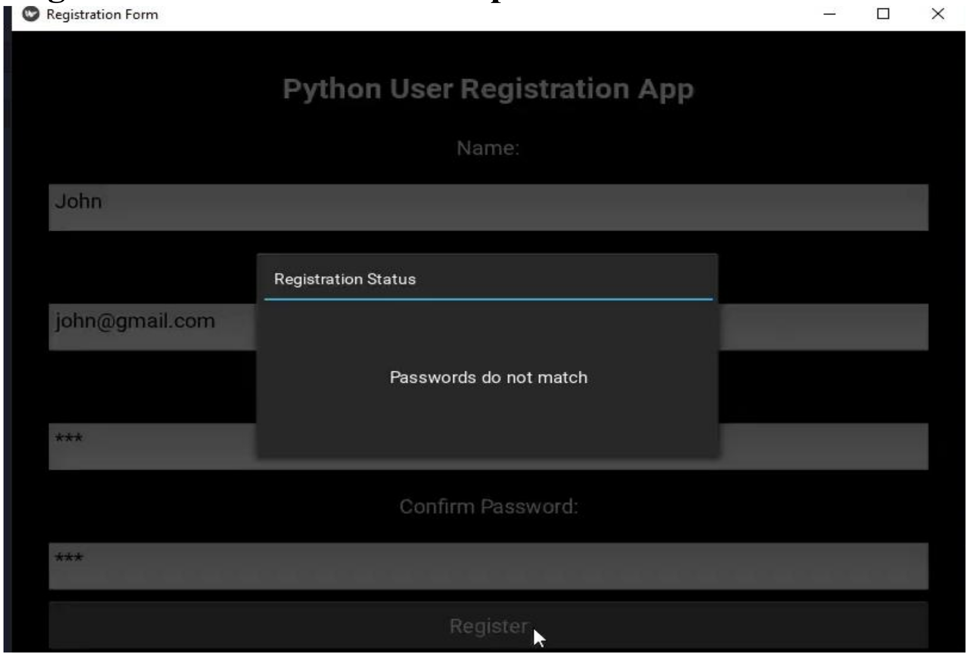


2. Registration Status: Unfilled fields



Please fill in all fields

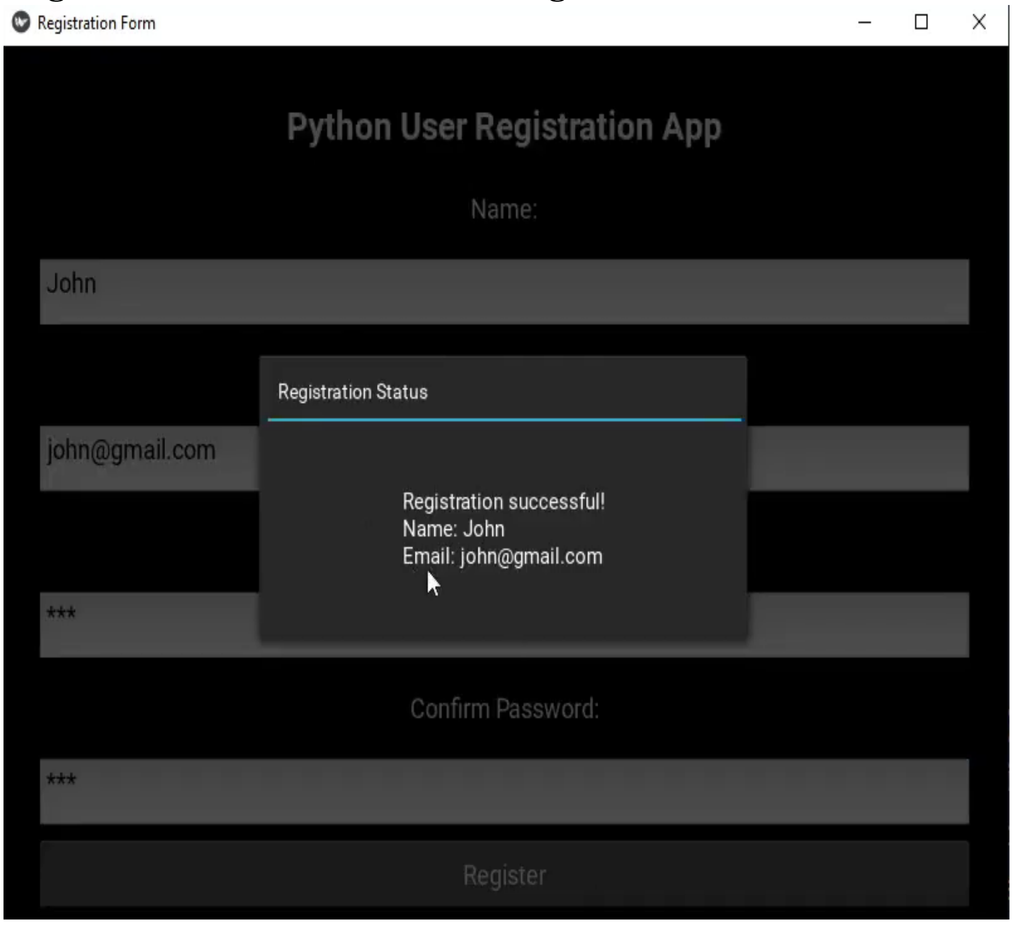
3. Registration Status: Invalid Inputs



Registration Status

Passwords do not match

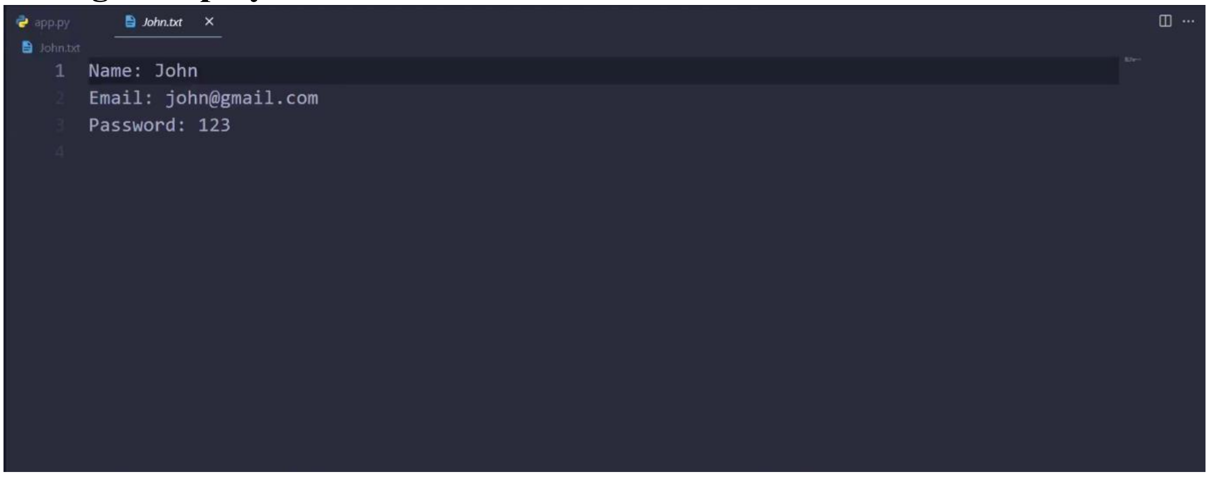
4. Registration Status: Successful Registration



Registration Status

Registration successful!
Name: John
Email: john@gmail.com

5. **Storage: Display stored data**



```
Name: John
Email: john@gmail.com
Password: 123
```

Testing Approach:

This project performs multi-stage testing approach:

- **Unit Testing:** It includes validator.py module. It tests function is_valid_email .
- **Integrated Testing:** It includes GUI + validation in addition with Validation + storage.

Challenges Faced:

- We have to maintain input validation for multiple fields, consecutively.
- We have to handle file operations correctly.
- We have to keep our codes well-structured.
- Our all the codes must be modular.
- We have to always ensure that the SQLite connection was opened, transactions were committed and the connection was closed correctly with every operation to prevent database locks or corruption.

Learnings and Key Takeaways

These types of simple projects for beginners provides them valuable experience in several key areas of application development:

- Gain in hands-on experience with Python kivy.
- Improved understanding of file handling and data storage.
- Enhanced knowledge of GUI event-driven programming.
- Experience about how separation of GUI, logic and data dramatically simplifies debugging and maintenance.
- Gain in proficiency in using Python's sqlite3 module to perform fundamental CRUD operations.

Potential Future Enhancements:

To expand and enhance the functionality and professionalize the application, the following features could be added:

1. **User Authentication (Login):** Add a separate screen that allows registered users to log in by validating their entered credentials against the stored database records.
2. **Password Hashing:** Implement robust security practices by using libraries like `bcrypt` or `hashlib` to store one-way hashed passwords instead of plain text.
3. **Data Retrieval and Display:** Add a feature within the application to display a list or table of all registered users (Read operation).

4. **Modern GUI:** Migrate the application from “kivy” to a more modern library like “Flet”, “PyQt”, :BeeWare”, etc. for enhanced aesthetics and complex layout management.

References:

- Python Standard Library Documentation (kivy, sqlite3).
- GeeksforGeeks Python GUI Tutorials.
- YouTube lectures for basic understandings of Python programming. (College Wallah).
- VITyarthi