Demo Account Registration Web Application

Group Name: Python Group (Demo email Account)
BCP104 - Programming with Python
Department of Computer Science
Accra Technical University

September 17, 2025

Group Members

Name	Index Number	Contribution
Nater-Tawiah Isaac Yohanes	01245954B	Full-stack development, validation logic, deployment
Charles Ebanyenle	01233698B	Full-stack development, validation logic, deployment
Osman Anwar	01244669B	No contribution
Darko-Ameyaw Joel	01245244B	No contribution
Gideon Nana Ofosu	01246061B	No contribution

Introduction

This project is a simple Python web application built using Flask. It allows users to create a demo account by submitting their full name, email, and password. The application securely stores user data in a SQLite database and performs server-side validation. It is deployed at:

https://tawiahisaac.pythonanywhere.com/

Problem Statement

Many beginner web applications lack proper validation, modularity, and deployment readiness. This project solves that by offering a clean, DRY, and secure registration system with real-world deployment.

Project Objectives

- Build a secure registration form using Python and Flask.
- Validate user input server-side.

- Store user data securely using password hashing and SQLite.
- Deploy the app online for public access.

Requirements

Functional Requirements

- User registration form
- Email and password validation
- Password hashing
- SQLite database integration

Non-Functional Requirements

- Usability: Simple and intuitive interface
- Performance: Fast response time
- Compatibility: Works on modern browsers

System Overview

The app consists of a single Flask route that handles both GET and POST requests. Users fill out a form, and the server validates and stores the data. Target users are students and developers learning Flask.

Technical Design

Architecture

- Frontend: HTML form embedded in Flask template
- Backend: Flask route with validation logic
- Database: SQLite for persistent storage

Main Components

- `register()`: Handles form rendering and submission
- `init_db()`: Initializes the database
- `save_user()`: Saves user data securely
- `is_valid_email()`, `is_strong_password()`: Validation helpers

Technologies Used

- Python 3.11
- Flask (micro web framework)
- SQLite (lightweight database)
- Werkzeug (for password hashing)
- HTML/CSS (embedded in Flask template)

Implementation Details

Core Logic

```
def is_valid_email(email):
    return re.match(r'^[\w\.-]+@[\w\.-]+\.\w+$', email)

def is_strong_password(password):
    return len(password) >= 8 and re.search(r'[A-Z]', password) and
       ↪ re.search(r'\d', password)

def save_user(full_name, email, password_hash):
    with sqlite3.connect(DB_PATH) as conn:
        conn.execute("INSERT␣INTO␣users␣(full_name,␣email,␣
           ↪ password_hash)␣VALUES␣(?,␣?,␣?)",
                     (full_name, email, password_hash))
```

Challenges Faced

- Ensuring all validation logic was modular and reusable
- Preventing duplicate email registration
- Deploying to PythonAnywhere with proper database setup

Testing & Evaluation

Manual testing was performed using various input combinations. The app correctly handles:

- Missing fields
- Invalid email formats
- Weak passwords
- Duplicate email entries

Conclusion & Future Work

This project demonstrates how to build a secure, modular, and deployable Flask web app. Future improvements include:

- Adding login and session management
- Integrating image upload
- Sending email confirmations

References

- Flask Documentation
- PythonAnywhere Hosting
- Werkzeug Security