**Agribot-ai :** **AgroBot Universal AI based Agricultural Assistant**

**1. Project Overview**

**AgriBot** is a web-based application designed to provide accessible and reliable agricultural advice to farmers. The core of the project is an intelligent chatbot that can answer farming-related questions. To ensure a secure and personalized experience, the application is built with a robust user authentication system that distinguishes between two primary roles: **Users** (farmers seeking information) and **Admins** (administrators who manage the application).

The project is built using a modern technology stack:

* **Backend:** Python with the Flask web framework.
* **Database:** SQLite, managed via the Flask-SQLAlchemy extension.
* **Security:** Password hashing is handled by Flask-Bcrypt.
* **Frontend:** HTML for structure, with CSS for styling.

This document details the architecture and functionality of the dual-login system, which is the foundational module of the AgriBot application.

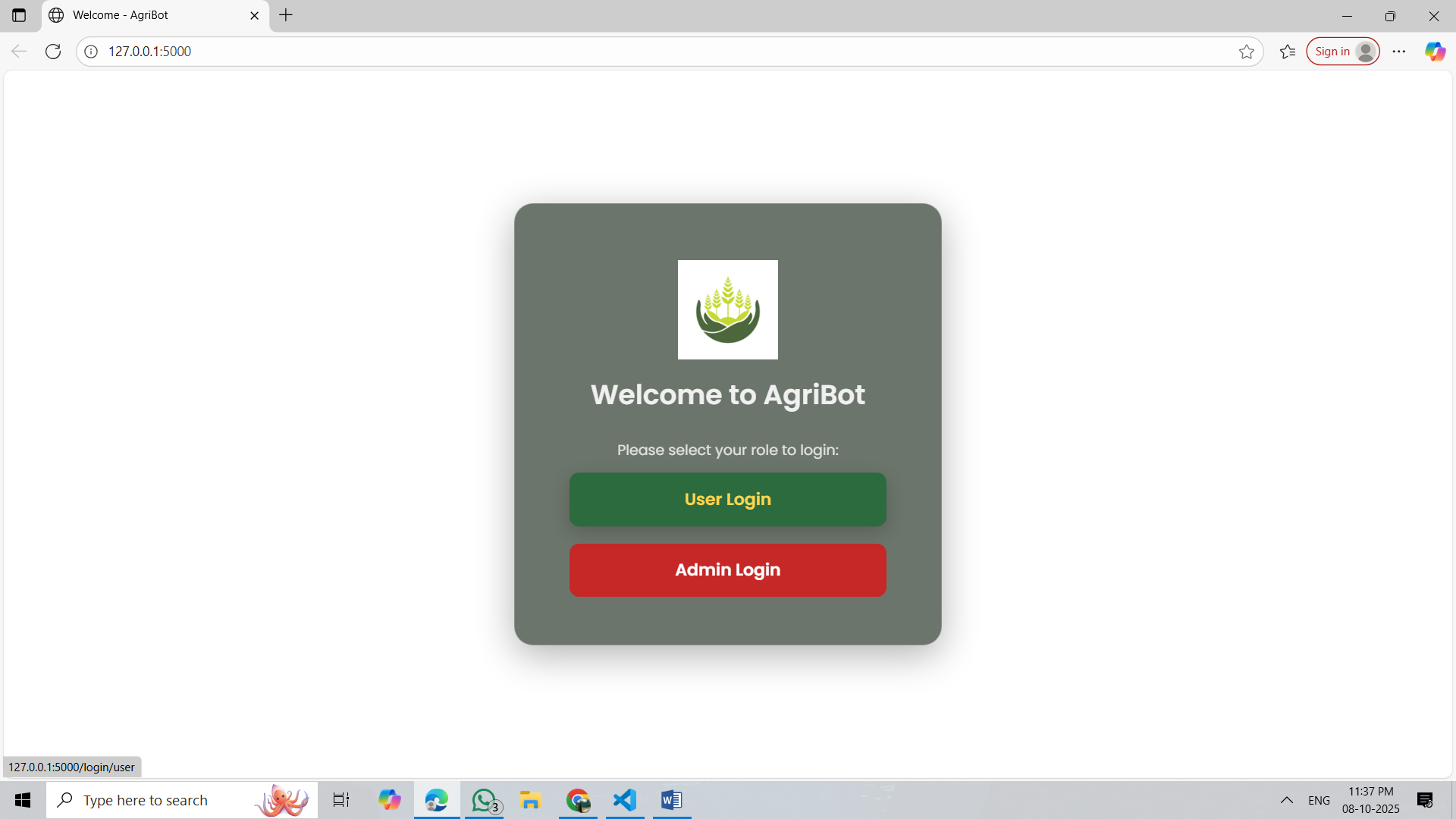
**2. How the Dual Login System Works**

The application features two distinct login pages to securely separate the user and admin environments. The workflow is designed to be intuitive and secure, guiding each user to their correct dashboard.

**a. The Role Selection Page**

The user's journey begins at a central landing page. This page is the main entry point and prompts the user to identify their role.

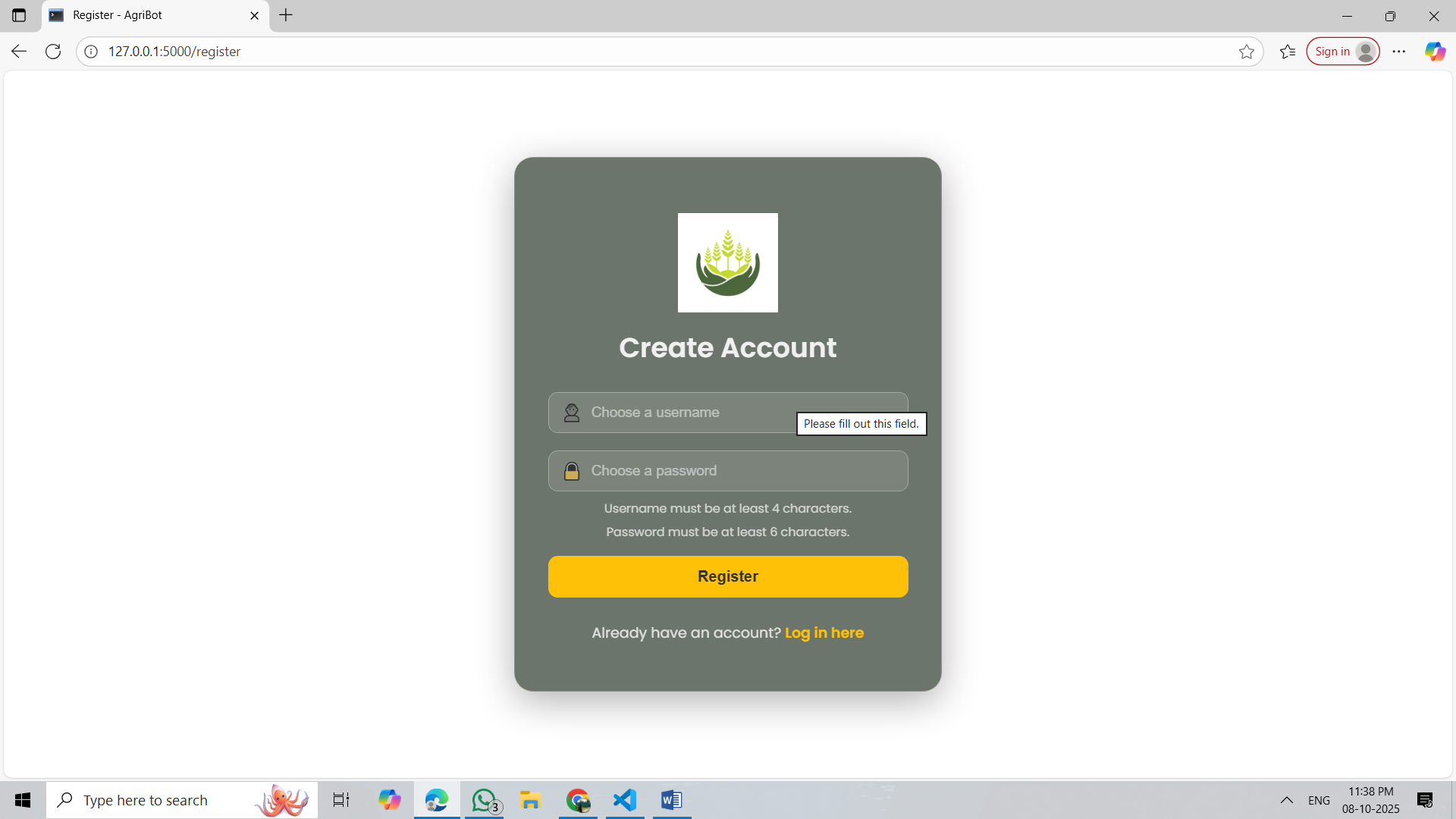
* **Functionality:** It presents two clear options: "User Login" and "Admin Login".
* **Technical Implementation:** This page is generated by the chooser() function in app.py, which renders the chooser.html template. Each button is a hyperlink that directs the user to the appropriate login route (/login/user or /login/admin).



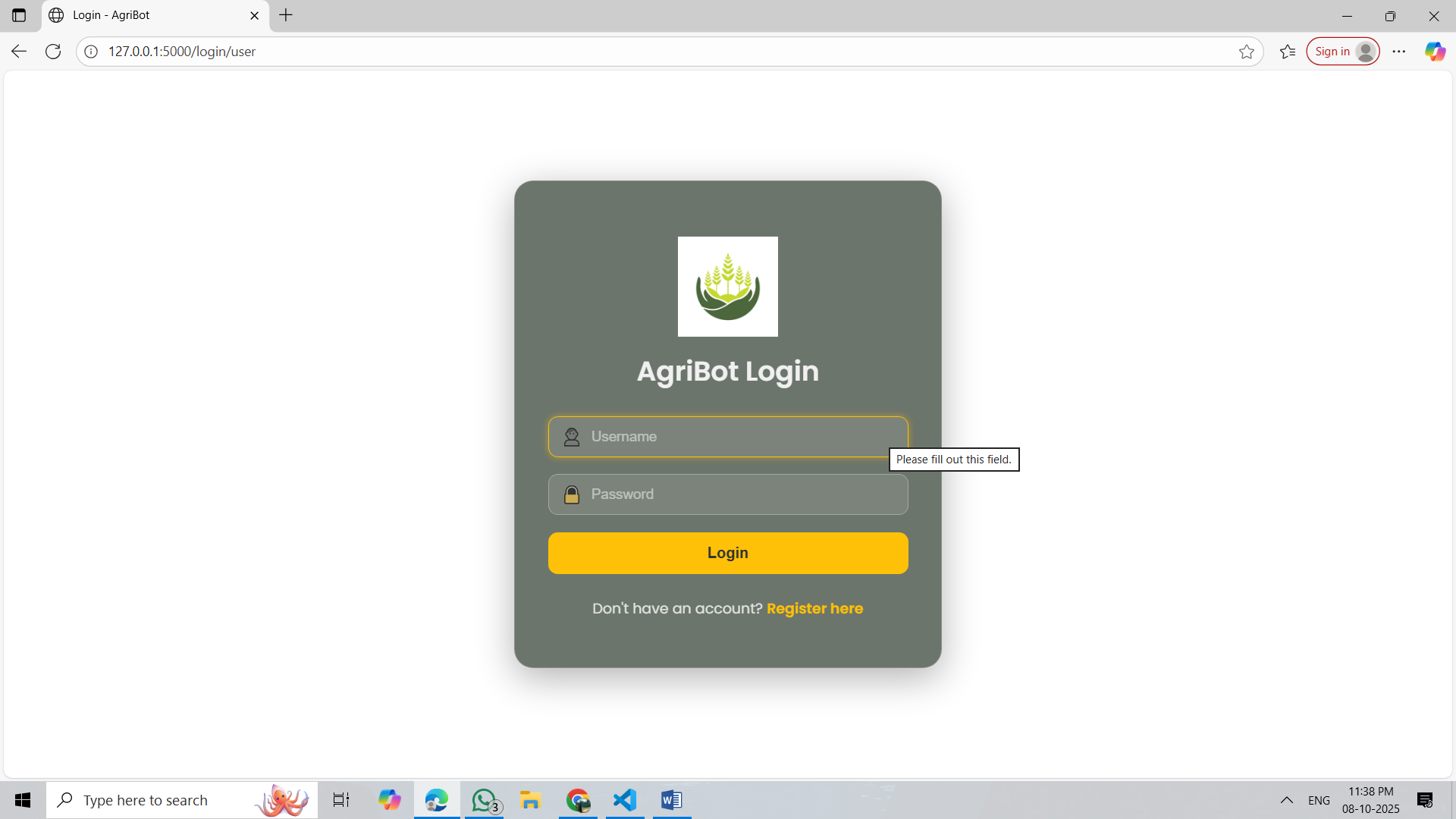
**b. User Login and Registration**

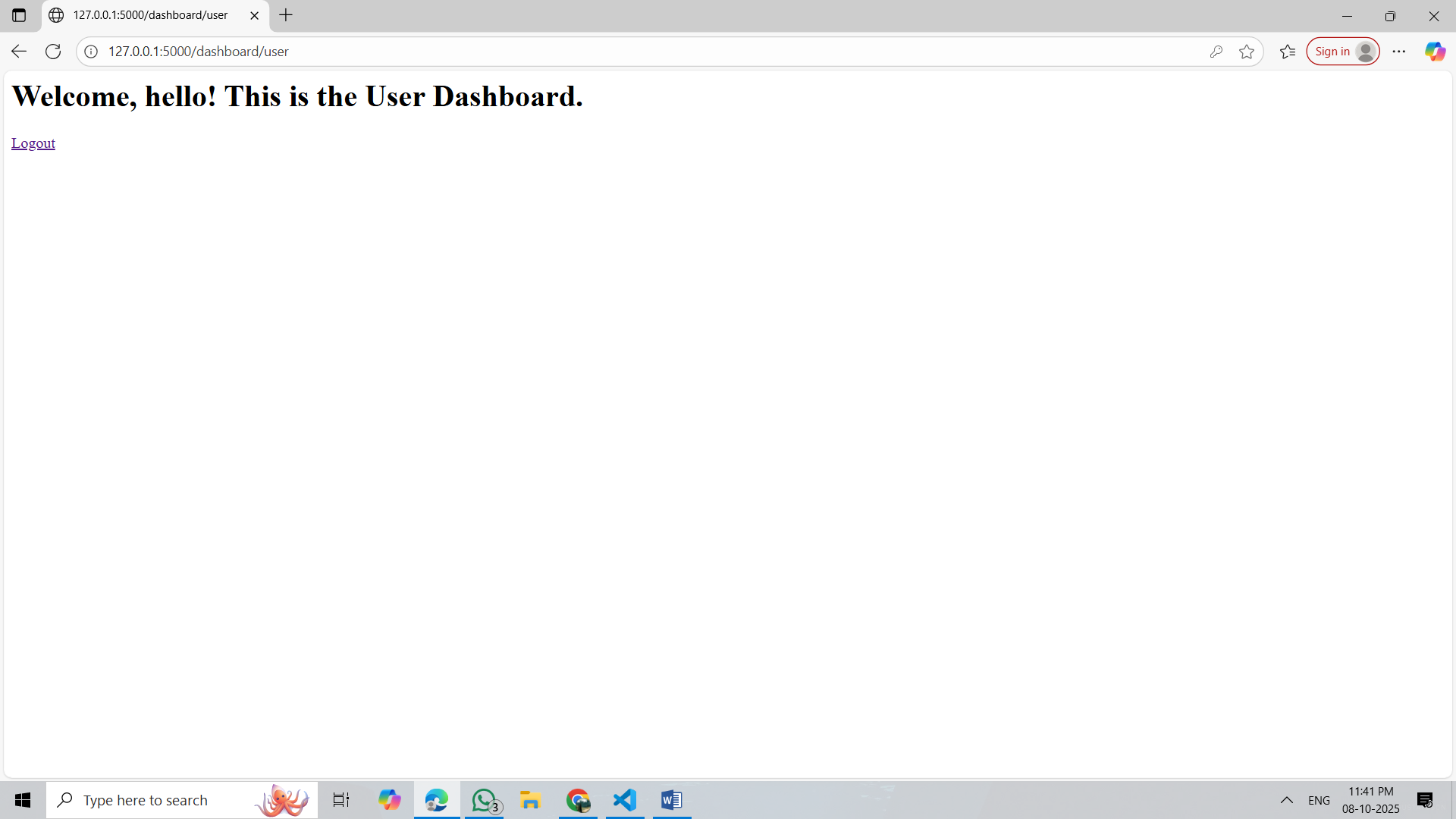
This flow is designed for the primary users of the application (farmers).

* **Registration:**
  1. New users click the "Register here" link on the user login page.
  2. They fill out a form with a username and password. The system enforces basic **constraints**: the username must be at least 4 characters, and the password must be at least 6 characters.
  3. When submitted, the register() function in app.py validates the data, checks if the username already exists, and if all constraints are met, it securely hashes the password and saves the new user to the database with the default role of 'user'.



* **Login:**
  1. Existing users enter their credentials on the user login page.
  2. The user\_login() function in app.py receives the form data.
  3. It searches the database for a user with the matching username **and** a role of 'user'.
  4. If a match is found, it uses **Bcrypt** to compare the submitted password with the stored hash.
  5. Upon successful verification, the user's ID, username, and role are stored in a **session**, and they are redirected to their personal dashboard.

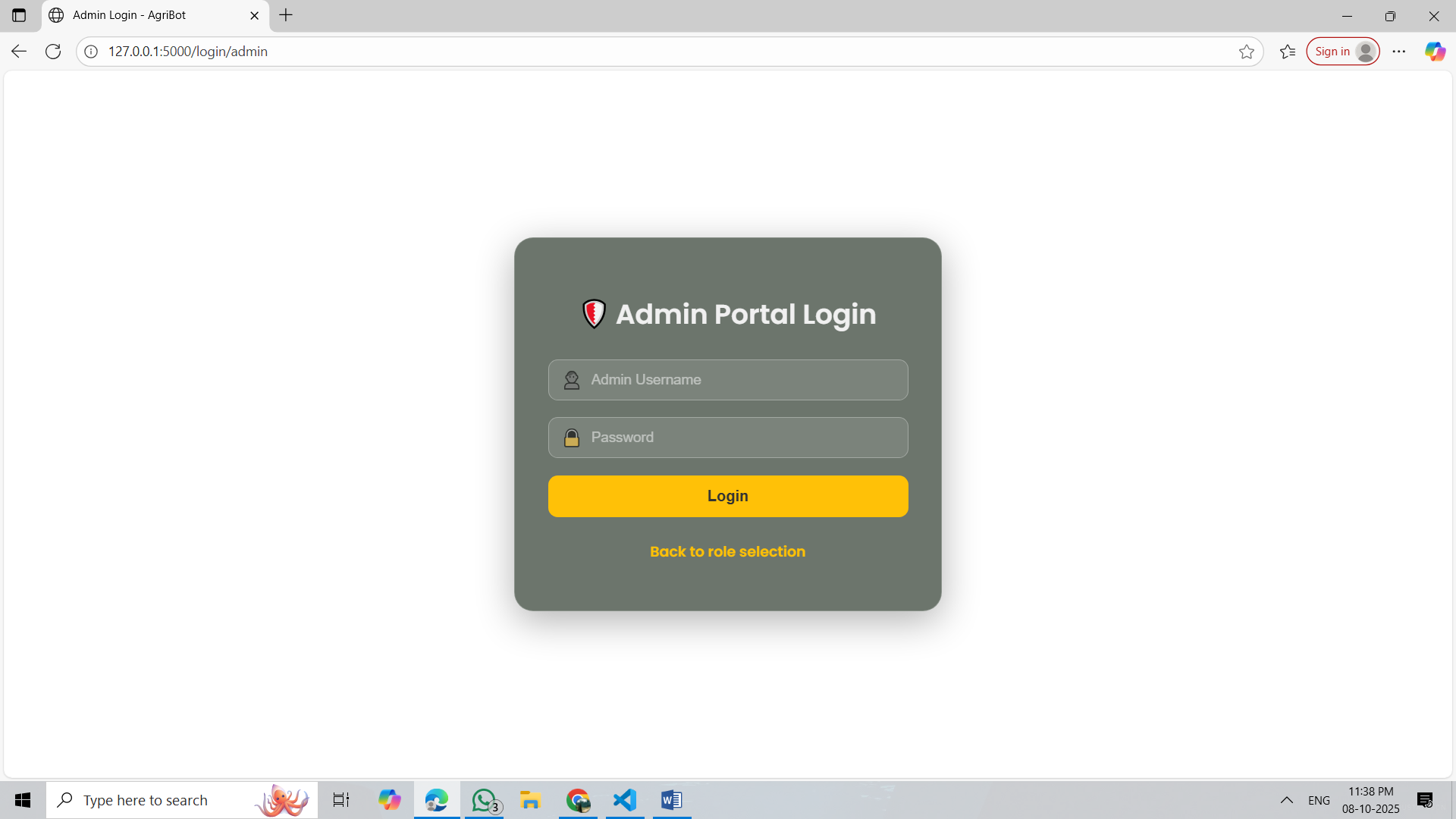


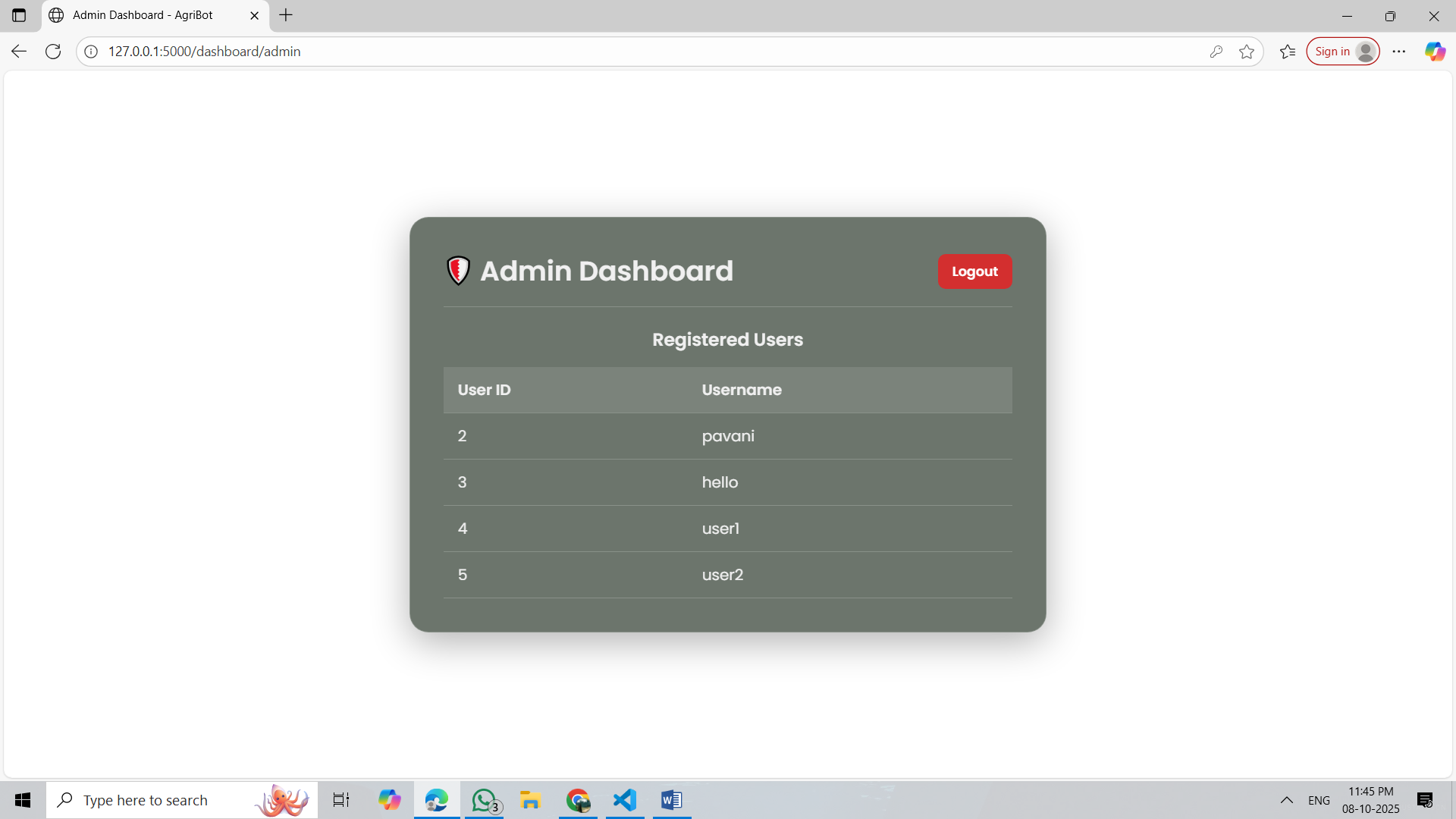


**c. Admin Login**

This is a separate and protected flow for application administrators.

* **Login Process:**
  1. An admin selects the "Admin Login" option from the main landing page.
  2. They are directed to a distinct admin login page (admin\_login.html).
  3. The admin\_login() function in app.py handles the form submission.
  4. Crucially, the database query specifically looks for a user with the matching username **and** a role of 'admin'. This ensures that a regular user cannot log in through the admin portal, even if they know the URL.
  5. Upon successful authentication, the admin's details are stored in the session, and they are redirected to the admin dashboard, where they can view data like the list of registered users.





Current default dashboard which gives information about usernames and user\_ids which can be used for future references by the admin.

**3. Database Integration and Security**

The database is the backbone of the authentication system, storing user credentials and roles securely and efficiently.

**a. The User Model**

The structure of the user data is defined in app.py using a Python class called User, which SQLAlchemy uses to create and manage the user table in the database. The model has three essential columns:

* id: A unique integer that serves as the primary key for each user.
* username: A unique string to identify the user.
* password: A string that stores the **hashed** password.
* role: A string that defines the user's permissions, either 'user' or 'admin'.

**b. Database Creation and Connection**

When the application is run for the first time (python app.py), the following happens:

1. The line app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///database.db' tells the application to create or connect to a SQLite database file named database.db.
2. The db.create\_all() command reads all the defined models (like our User class) and automatically creates the corresponding tables in the database file if they don't already exist.
3. The application also automatically creates a default **admin** account (username: admin, password: admin123) on its first run to ensure the admin portal is immediately accessible.

**c. Password Security with Bcrypt**

Storing passwords in plain text is a major security risk. This application uses the **Flask-Bcrypt** library to ensure password security through a process called **hashing**.

* **During Registration:** When a user signs up, their chosen password is not saved directly. Instead, it is passed through bcrypt.generate\_password\_hash(). This function converts the password into a long, complex, and irreversible string (the hash). This hash is what gets stored in the database.
* **During Login:** When a user tries to log in, the application does not un-hash the stored password. Instead, it takes the password they just typed and hashes it using the same algorithm. It then compares this newly generated hash with the one stored in the database. The bcrypt.check\_password\_hash() function handles this comparison securely. If the hashes match, the password is correct.

**🚀 How to Run AgriBot:**

A step-by-step guide to get the AgriBot application running locally.

**1. Setup**

Clone the repository and navigate into the project directory.

Bash:

* git clone <your\_repository\_url>
* cd <project\_folder\_name>

**2. Environment & Dependencies**

Create a virtual environment and activate it.

Bash:

# Create environment

python -m venv venv

# Activate (Windows)

.\venv\Scripts\activate

# Activate (macOS/Linux)

source venv/bin/activate

Install the required packages.

Bash:

pip install Flask Flask-SQLAlchemy Flask-Bcrypt

**3. Run the App**

Start the Flask server. This will also create the database.db file automatically.

Bash:

python app.py

Open your browser and go to [**http://127.0.0.1:5000**](https://www.google.com/search?q=http://127.0.0.1:5000).

**[Admin Access**

A default admin account is created automatically.

* **Username:** admin
* **Password:** admin123]