Brief Explanation of the Approach Taken and Challenges Faced

**Approach**

1. Modular Design: The script is designed in a modular fashion with distinct functions for each functionality:

- hash\_password(password): Hashes passwords using SHA-256 for secure storage.

- load\_credentials(): Loads user credentials from a JSON file.

- save\_credentials(credentials): Saves user credentials to a JSON file.

- validate\_password(password): Validates passwords against standard security criteria.

- register\_user(username, password): Registers a new user by validating and storing their credentials.

- login\_user(username, password): Authenticates users by validating their credentials.

- main(): Provides the user interface for registration and login.

2. Password Security: Passwords are hashed using SHA-256 before storage to enhance security.

3. Password Validation: During registration, passwords are validated to ensure they meet security criteria:

- At least 8 characters long

- Contains at least one uppercase letter

- Contains at least one lowercase letter

- Contains at least one digit

- Contains at least one special character

4. Case-Insensitive Usernames: Usernames are converted to lowercase during both registration and login to prevent case sensitivity issues.

5. JSON for Storage: User credentials are stored in a JSON file (user\_credentials.json) for simplicity and ease of access.

6. Error Handling: The script includes basic error handling to manage common issues such as:

- Username already exists

- Username not found

- Invalid password

**Challenges Faced**

1. Password Security and Validation: Ensuring passwords are stored securely and meet specific security criteria was a primary challenge. Implementing SHA-256 hashing and regular expression-based validation helped address this.

2. Case Sensitivity: Preventing case sensitivity for usernames required careful handling during both registration and login. This was resolved by converting all usernames to lowercase.

3. Modularity and Readability: Writing clean, readable, and well-structured code while ensuring all functionalities are modular and reusable was essential. This required careful planning and organization of functions and their responsibilities.

4. Error Handling: Implementing comprehensive error handling to manage various user input errors and edge cases, such as existing usernames and invalid passwords, was crucial for robustness.

5. JSON Storage Management: Managing user credentials using a JSON file involved ensuring data integrity and proper handling of file operations. Loading and saving credentials efficiently while maintaining readability was key.

**Github Link:**

https://github.com/Nwafor6/Payfly\_Test

**Running the Code**

To run the user authentication system, follow these steps:

1. Install Python: Ensure you have Python installed on your machine. You can download it from python.org (https://www.python.org/).

2. No Additional Packages Required: This script does not require any additional Python packages beyond the standard library.

3. Clone the repository: Clone the repository into your machine.

4. Run the Script: Open a terminal or command prompt, navigate to the directory where auth\_system.py is saved, and run the following command:

bash

python auth\_system.py

5. Follow the Prompts: The script will prompt you to register a new user, log in with an existing user, or exit the program. Follow the on-screen instructions to interact with the authentication system.

Example Usage

1. Register a New User:

- Choose the "Register" option.

- Enter a username (which will be converted to lowercase).

- Enter a password that meets the validation criteria (at least 8 characters long, contains uppercase, lowercase, digit, and special character).

2. Log In:

- Choose the "Login" option.

- Enter the username (case-insensitive).

- Enter the password.

3. Exit:

- Choose the "Exit" option to terminate the script.