Exercise work

Exercise work is “User log in” console program that allows user to create new user with user information and to modify them after sign in. All user inputs are sanitized to maximize security of program. Program runs from “main.py” and it needs “users.json” and “userdata.json” files. These two json files should contain “users” and “userdata” items that are filled by using program.

When program is started it ask if you are new user. If input is “y” user creation starts, else it will move to ask if you want to log in. Again “y” will start login processes and “n” will lead to system turn off question. Here “y” will lead to system shutdown and “n” starts program from start.

Creating new user follows these steps:

- Username
    - “Enter username” input
    - Checks if inputted username is not already used from users.json.
    - If available moves to next step, if not repeats “enter username”
- Password
    - “Do you want system to generate new password for you” input.
        - Y = program generates password and shows it.
        - N = “Enter password” input.
    - “Type password again” input:
        - If match moves to next step.
        - If it does not match repeat from start of password.
- Password is hashed.
- User information:
    - Name input (text sanitization).
    - Age input (number sanitization).
    - Sex input (text sanitization).
    - Address input (text sanitization).
    - Email input (email sanitization).
    - User id generated using uuid4.
- “User” and “userd” items are created and filled with given information.
    - User:
        - User id.
        - Username.
        - Hashed password.
    - Userd:
        - User id.
        - Name.
        - Age.
        - Sex.
        - Address.
        - Email.
- User is added to users.json.
- Userd is added to userdata.json.

Login as user follows these steps:

- Username input.

- Password input.
- Authentication.
    - Check if username exists in users.json.
        - If exists take hashed password and user id from users.json.
        - Else authentication fails.
    - Check if passwords match.
        - If matches, return user id.
        - Else authentication fails.
- If authentication fails.
    - Try again input:
        - Y = start of login, tried username is added to triedUserName dict.
        - N = exit log in.
- If authentication was successful
    - Display user information from userdata.json using user id from authentication

Display user follows these steps:

- Prints all user information
- “Do you want to update” input:
    - If y starts update step.
    - If n exits from display to back to main.
- Update steps:
    - Uses update function to go through all information and what to update.
    - Old user data in userdata.json is replaced with new information.
    - Starts display user function again if user still wants to update some information.

Input sanitization:

- Yn (only allows upper or lowercase “y” and “n”).
- Usr (only allows characters and numbers).
- Pass (uses getpass. only allows characters, numbers, and specific special characters. Length needs to be between 8 and 50 and must contain one of lower, upper, number, and special character).
- Text (only allows characters, numbers, and specific special characters).
- Email (only allows characters, numbers, and specific special characters. There needs to be one “@” character in email).

Secure programming solutions

- Password hashing.
    - Uses hashlib.pbkdf2_hmac() function with ‘sha512’ to ensure password is almost impossible to guess.
- Password requirements.
    - Length requirements.
    - It must include at least one lower-, uppercase, number, and special character.
- User input sanitization.
    - Length is capped.
    - Using whitelist to allow only specific characters. There are multiple different whitelists for different sanitization types.

Testing

Testing was done using only manual testing. I started to make automated tests, but I have not made test that use user inputs before in this scale. From testing I implemented three strike system for log in. If user fails to log in using account 3 times system is turned off.

Due to strict sanitization user inputs should stop most if not all abuses or attack vectors from user inputs to system. This does lead to unwanted restriction in passwords where there could be more special characters usable without raising the risks to system.

Missing implementation

I was planning to encrypt userdata.json file to make it unreadable unless you had key to decrypt it, but I had technical problems and I did not have enough time to make it work.

Security issue

Passwords can be brute forced with time even though passwords are encrypted. This could be made even harder with time locks to accounts in case of too many failed attempts.

Improvements

Program would be better fit to course if it were web based since there would be more security issues/problems to fix and there would be more options for secure programming solutions. More robust email validation to make sure email is legit. Time lock/delay to account if there are too many failed logins. Less strict whitelists for sanitization, since currently they might hamper some emails and passwords people use. GUI version of program.