Assignment 6: Databases

Question 1: Consider the bank application from Lab 5:

a- Create a function to return the transaction history for a given user. The function should have the following signature:

```
fn get_transactions_history(&self, u_name:&str)->Result<(),UBaseErr>
```

Using this function, you should be able to view the transaction history for a given user. A sample output should look like:

```
Matt received $140 from Jim on 12/10/2019 11:34 p.m.
Matt sent $100 from Andrew on 24/11/2019 01:12 p.m.
```

b- Edit the pay function, so it checks for the balance of a given user before allowing them to *pay* money to another user. The function should not allow the user to send money unless they have enough money in their bank account! That is, they must still have a positive balance after the transfer. (Hints: You may need to define a function *get_balance* in this task. You may also need to create an extra field in the users database to hold the users balance. This balance can be initialized when creating a user.)

c- Provide the necessary test functions to test your code in a and b.

Please upload your submission as a complete Rust project with the name "**Assignment6Question1**". The project must contain the database file in the *data* folder.

Question 2: Complete your bank project by creating a command-line app for accessing the database. The app should allow the user to send commands as arguments. Then, the app must be able to interpret these arguments and interact with the user properly.

Here are a set of arguments that the app must be able to work with:

1- **new:** This is a command to create a new user, the second, and the third arguments, in this case, should provide the username and password. If the user forgot to provide the necessary arguments, the app must prompt him/her to complete the command with whatever needed.
2- **transfer**: This is a command to send money from one user *(from_user)* to another user *(to_user)*. Please note that this function requires providing the password of the *from_user*. Therefore, the app must ask the user to enter his password to complete the payment operation.
3- **balance**: This is a command to show the balance of a given user. Again, this is a command that requires the user to enter his password first to be able to check his balance.

Here is a sample of what should the program look like. Please, note that the user input is highlighted in boldface.

```
# cargo run new matt mattpw
Adding user matt with password mattpw…
Operation done successfully!
# cargo run transfer steve matt 150
Please input your password:
```

```
stevepw
Sending money from steve to matt…
Operation done successfully!
Cargo run balance matt
Please input your password:
mattpw
Balance is $100
Operation done successfully!
```

Please upload your submission as a complete Rust project with the name “**Assignment6Question2**”. The project must contain the database file in the *data* folder.

Question 3: You have been using bcrypt to encrypt and validate the user password. However, bcrypt is getting old. Malicious users can now successfully attack bcrypt using massive concurrent attacks. Hence, best practice is to use argon2 instead (https://docs.rs/rust-argon2/0.5.1/argon2/). Rewrite the code to follow best practices (i.e. use argon2 instead of bcrypt).

Please upload your submission as a complete Rust project with the name “**Assignment6Question3**”. The project must contain the database file in the *data* folder.