No.	Security Issue	Java File	Possible Vulnerabilities	Solution Implementation
1	Unmasked password registration fields.	Login.java Register.java	Shoulder surfing when a user is typing his/her password: - Stolen identity - Unauthorized access	Password fields needs to be masked to prevent the vulnerabilities mentioned. Change from using JTextFields into JPasswordFields which cover the characters with a block character. Additionally, the password is held on this entry field as an array of characters instead of a string.
2	No input validation on login input fields	Login.java	Inputs are not checked such that the entered credentials are not screened for allowable entries. Unsanitized or invalidated inputs can be susceptible to: - SQL Injections - Buffer overflow - Null byte injection	Inputs for login must be validated such that the following are checked: 1. Uppercase letters (Password): A-Z 2. Lowercase letters (Username & Password): a-z 2. Numbers (Username & Password): 0-9 3. Symbols (Username): _-. 4. Symbols (Password): ~`!@#$%^&*()_-+={[}]|\:<,>.?/ Add functions that will check the login inputs (username and password) if it meets the minimum required inputs and won't allow SQL queries to be performed (', ", ; are not allowed which are found on SQL query strings) once login button is clicked. This function will dictate whether it will allow the login procedure to proceed any further or not.
3	Passwords not hashed	SQLite.java DB contents	Passwords are stored and accessed in the DB in plaintext: - Unauthorized access (should a known user be available) - Stolen password (can bruteforce users that match the password)	Passwords must be hashed prior to writing and upon reading a password query. SHA-256 & AES are used for hashing and encryption respectively. 1. Implement hashing at a controller level for both password verification (read) and password saving (write). 2. Update the current contents of the database such that the passwords are now in its hashed form.

4	No actual user authentication at login (i.e., if username & password match)	Login.java	Given credentials are not necessarily checked for user authenticity. This makes it susceptible to: - Unauthorized access	Entered validated/sanitized input credentials must be checked for matches on the DB which will dictate if the user will be authenticated or not. Implement controls that would verify the username and password before proceeding to the main screen should it be successful.
5	No logging of login attempts	Login.java SQLite.java	User login attempts (successful or not) are not being logged which could be USEFUL for: - Username attack detection (e.g., username enumeration) - Password attack/failure detection	Any login attempt must be logged with the time and who attempted to login (whether user is valid/found or not). Implement logging function on Login page, call and log accordingly whenever it meets certain conditions: - Successful Login - Failed Login - Locked Account Attempted to Login - Account Locked due to Failed Login Attempt - Login attempt on unknown username
6	No invalid/error login pop-ups/notifiers	Login.java	There is a need to notify user of failed login attempts should an error occur or credentials given are invalid. Should this occur, the user should be notified so in a manner that is ambiguous or generic. Failure to do so could make the program vulnerable to: - Username and password enumeration - Stack trace leak (which can give information about the inner workings of the program)	Warning or error pop-up messages should contain messages that are ambiguous to the user such that the error is generalized with no clue regarding invalidity of username and/or passwords will be shown.

7	No lock-out mechanism	Login.java SQLite.java	Repetitive unsuccessful login attempts should not be allowed and does not stop user from logging in after certain retries at a certain time. This makes it vulnerable to: - Brute force password attacks Note the ff.: 1. By this point, a known username must have been known by the malicious actor already. 2. Unlocking the account is recommended to be done with user (administrator) intervention. The administrator must also restore the user account's original role number as unlocking an account reverts the account to a client role.	An account lockout should be made immediately once the account reaches 8 consecutive attempts such that the account of the user will be set as Role 1 and the locked value to 1. This event will also be logged. For Login.java, assuming that inputs are validated, add and call a function that will check if the user account specified is valid (i.e., existing) and is unlocked. Once that it is verified, the succeeding login procedure will continue as normal, if not then the user will be warned that the "Logging in is not available at the moment." For Login.java, should the user be unlocked but was attempted to incorrectly logged in at least 8 times, then the lockout procedure shall commence in order to secure the account. For SQLite.java, lockout related functions are added that will enable lock and unlock functionality on a given account.
8	Exposed and incorrectly placed function	Register.java Frame.java	A function supposedly in Register.java was found to be misplaced in Frame.java which all functions that inherit Frame.java could call. This function contains code that can manipulate contents of the DB as it relates to user account registration. This security issue makes the program vulnerable to: - Accidental Exposure (from programmer perspective) - Function misuse	Move Frame.registerAction() to Register.registerAction()

9	No input validation on register input fields	Register.java	User credentials inputs are not validated making it vulnerable to: - SQL Injections - Buffer overflow - Null byte injection Minimum password requirements are also not being checked making it vulnerable to: - Weak passwords User account availability is not also checked making it vulnerable to: - Stolen identity - Unauthorized access	Inputs for user account credentials must be validated such that the following are checked: 1. Uppercase letters (Password): A-Z 2. Lowercase letters (Username & Password): a-z 2. Numbers (Username & Password): 0-9 3. Symbols (Username): _-. 4. Symbols (Password): ~`!@#$%^&*()_-+={[}]|\:<,>.?/ Once the inputs are pre-sanitized, the following must be also met in order to proceed further on the registration process: 1. Minimum password length of 6 2. At least 1 uppercase letter, 1 lowercase letter, 1 number, 1 symbol 2. Password and confirm password contents match 3. Username is available Add functions that will (as it will dictate whether it will allow the register procedure to proceed any further or not): 1. Check the registration inputs (username, password, and confirm password) if it meets the minimum required inputs 2. Won't allow SQL queries to be performed (', ", ; are not allowed which are found on SQL query strings) once register
10	User account registration logging	Register.java	Registration attempts should also be logged which could be USEFUL for: - Auditing registered accounts (comparing registered on DB to registered by log) - Detecting possible attempts on user enumeration	Solving this would require implementing logging functionality on registration whether it is successful or not.