Security Issues Documentation Table

#	Security Issue	Java File to be Modified	Possible Vulnerabilities if Not Addressed	Solution to be Implemented
Sample	Unmasked password registration fields	Register.java	Shoulder surfing when a user is typing his/her password: - Stolen identity - Unauthorized access	Password needs to be masked to prevent the vulnerabilities mentioned Change from `password = new javax.swing.JTextField();` to `password = new javax.swing.JPasswordField();`
1	Unmasked password field in login	Login.java	Shoulder surfing attack: - Password visible to onlookers - Credential theft - Unauthorized system access	Change password field from `passwordFld = new javax.swing.JTextField();` to `passwordFld = new javax.swing.JPasswordField();`
2	No authentication logic	Login.java	Complete bypass of security: - Anyone can access the system - No credential verification - Unauthorized access to all roles	Implement authentication in `loginBtnActionPerformed()` method to verify credentials against database before calling `frame.mainNav()`
3	Unmasked confirm password field	Register.java	Shoulder surfing during registration: - Password confirmation visible - Registration credentials exposed - Account compromise before first use	Change confirm password field from `confpassFld = new javax.swing.JTextField();` to `confpassFld = new javax.swing.JPasswordField();`
4	Password confirmation not validated	Frame.java	Password mismatch errors:	Modify `registerAction()` method to compare password and

#	Security Issue	Java File to be Modified	Possible Vulnerabilities if Not Addressed	Solution to be Implemented
			- Users may enter different passwords - Account lockout due to forgotten password - Poor user experience	confpass parameters before calling `addUser()`
5	No input validation for empty username	Login.java	Authentication bypass attempts: - Empty username submissions - Database errors - Potential SQL injection	Add validation in `loginBtnActionPerformed()` to check if `usernameFld.getText()` is not empty before proceeding
6	No input validation for empty password	Login.java	Security bypass attempts: - Empty password submissions - Weak authentication - System compromise	Add validation in `loginBtnActionPerformed()` to check if `passwordFld.getText()` is not empty before proceeding
7	No account lockout mechanism	Login.java	Brute force attacks: - Unlimited login attempts - Password guessing attacks - Automated credential stuffing	Implement failed login counter and temporary account lockout after 3-5 failed attempts
8	No input validation for registration fields	Register.java	Invalid user creation: - Empty usernames/passwords - Database integrity issues - Account creation failures	Add validation in `registerBtnActionPerformed()` to ensure all fields are filled before registration
9	No password complexity requirements	Register.java	Weak passwords: - Easy to guess passwords - Dictionary attacks - Account compromise	Implement password policy checking (minimum length, uppercase, lowercase, numbers, special characters)
10	No username format validation	Register.java	Invalid usernames: - SQL injection via special characters - Display issues - Authentication problems	Validate username to allow only alphanumeric characters and specific length (e.g., 4-20 characters)

#	Security Issue	Java File to be Modified	Possible Vulnerabilities if Not Addressed	Solution to be Implemented
11	SQL injection in user registration	SQLite.java	Database compromise: - Data theft - Data manipulation - Complete system takeover	Replace string concatenation in `addUser()` method with PreparedStatement as shown in commented code (lines 189-194)
12	SQL injection in user lookup	SQLite.java	Information disclosure: - User enumeration - Password extraction - Privilege escalation	Implement secure authentication method using PreparedStatement instead of string concatenation
13	Passwords stored in plain text	SQLite.java	Catastrophic data breach: - All passwords exposed if database compromised - No defense in depth - Regulatory compliance violations	Hash passwords using BCrypt or similar before storing in `addUser()` methods
14	No feedback on registration result	Register.java	Poor user experience: - Users unsure if registration succeeded - Duplicate registration attempts - Username conflicts	Add success/error messages before navigation in `registerBtnActionPerformed()`
15	Silent exception handling	SQLite.java	Hidden security errors: - SQL injection attempts not logged - Authentication failures not tracked - Debugging difficulties	Add proper logging in catch block at line 278 in `getUsers()` method
16	Hardcoded weak default passwords	Main.java	Default credential attacks: - Known passwords in source code - Easy system access - Backdoor vulnerability	Remove hardcoded passwords (lines 58-62) or use strong, unique passwords that must be changed on first login
17	Password retrieval in getUsers()	SQLite.java	Information disclosure: - All passwords exposed to any code - Internal threats - Privilege escalation	Modify `getUsers()` method to exclude password field from SELECT statement or return hashed values only

#	Security Issue	Java File to be Modified	Possible Vulnerabilities if Not Addressed	Solution to be Implemented
18	No logging of authentication events	Login.java	No audit trail: - Cannot detect attacks - No forensic capability - Compliance issues	Add logging for successful/failed login attempts using the existing logs table
19	All role buttons visible	Frame.java	Information disclosure: - System structure exposed - Social engineering attacks - Unauthorized access attempts	Hide role-specific buttons based on authenticated user's actual role
20	No session management	Frame.java	Unauthorized access: - No user context tracking - Cannot enforce access control - Multiple user confusion	Implement session management to track logged-in user and their role throughout the application