ABSTRACT

This project presents an implementation of a basic Login and Registration System in C++ designed as a command-line interface to simulate user authentication for applications or websites. This system provides an introductory model for handling user credentials with essential validations, making it suitable for foundational learning in user authentication workflows or prototyping basic authentication features. The system’s main functionalities include user registration, input validation for security, email verification, and login. Through a series of validation functions, the program ensures user-provided data meets specified requirements, thereby safeguarding the integrity of data entered and adhering to basic security standards.

The user registration process is the primary component of this system. It prompts new users to enter personal information, including a first name, last name, email address, and a password. This information undergoes a series of validation checks designed to enforce input correctness and security. For example, the first name and last name are required to start with uppercase letters, as per typical naming conventions, to establish a standard for user records. The email validation function is more intricate, verifying that the provided address includes exactly one "@" symbol and one dot (.), while ensuring these elements are in a logical order (i.e., "@" precedes the dot). This function also restricts the email from starting with a number or other non-letter characters, ensuring it follows a standard email format.

Password validation is particularly critical in this system, as passwords often represent the primary line of defense against unauthorized access. The system enforces a length requirement, where passwords must be between 8 and 15 characters to balance security with user convenience. Furthermore, passwords must meet complexity criteria, requiring at least one uppercase letter, one lowercase letter, one digit, and one special character from a specified set (@, #, \_). This combination helps prevent weak passwords that are vulnerable to guessing attacks and encourages users to adopt stronger, more complex passwords. The system checks for spaces in the password, as passwords with spaces may lead to parsing issues or potential security concerns in certain systems.

After completing registration, the system simulates an account verification step, mimicking a two-factor authentication mechanism commonly found in modern applications. A random 4-digit verification code is generated, representing a code sent via email for account confirmation. This code is printed to the screen to simulate email delivery, as actual email functionality is not included in this basic implementation. Users must input the verification code to confirm their identity and activate their account. Successfully entering this code allows the user to proceed, verifying their account and moving them to the login stage. This verification process adds an extra layer of security, demonstrating the importance of confirming identity beyond mere password protection.

The final step in the process is the login function. Here, users enter their username and password to access their account. Once the correct credentials are entered, the system confirms login success. However, because this is a prototype, there is currently no actual storage or retrieval of user credentials beyond the program’s runtime, so login verification is limited to the same session. Expanding this system to store credentials in a file or database would allow for persistent user data across multiple sessions, enabling users to register, log out, and log in again with the same credentials. Additionally, storing credentials securely (e.g., by hashing passwords before storage) would align this system with best practices in authentication security.

While this project provides a structured entry point to understanding authentication systems, it also highlights some limitations and areas for enhancement that would make it suitable for production-level use. Potential improvements include incorporating a database or file system to store and retrieve user data, which would enable the system to verify login credentials against previously stored records. Furthermore, implementing password hashing would protect passwords in storage, ensuring they are not vulnerable to unauthorized access or breaches. Additional measures could include duplicate checks to prevent multiple registrations with the same username or email and more advanced input sanitization to protect against potential injection attacks.

Error handling could be enhanced to manage edge cases, such as empty inputs or invalid characters, providing users with informative feedback to guide them toward successful registration. Implementing robust error messages not only improves user experience but also assists users in resolving input errors quickly, leading to smoother interaction with the system. Adding functions to handle password reset requests and implementing session management would further improve the system’s utility and security. These features, though outside the scope of the current implementation, represent important steps toward a fully functional, secure, and scalable authentication system.

Overall, this Login and Registration System demonstrates essential principles of user input validation, secure authentication design, and user experience flow, offering a solid foundation for future expansion. By building on this base, developers can incorporate additional security features and persistence mechanisms to create a more comprehensive, reliable, and secure user authentication system suitable for real-world applications.

**TEAM MEMBERS**

Jyoshna – AP23110011145

Prraneeth arraveeti – AP23110010713

Mokshagna – AP23110011110

Praneeth – AP23110011079