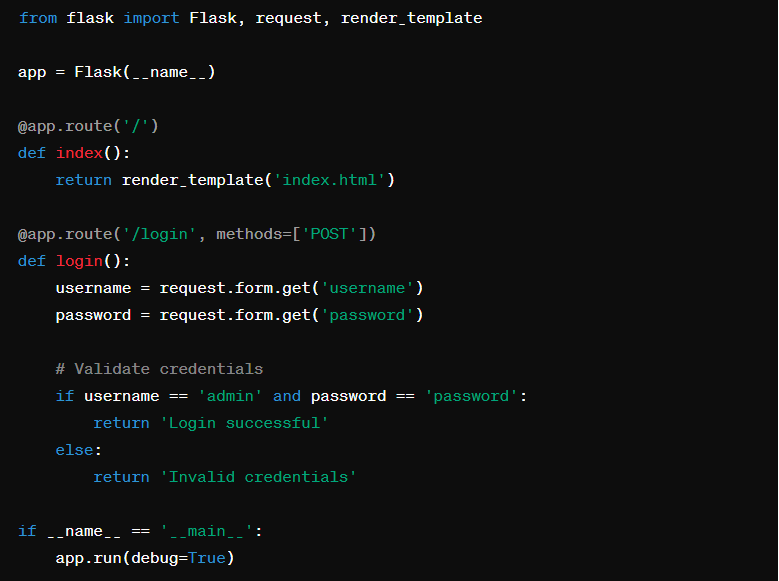
Task 3 : Secure Coding Review

-SHASHANK SS

Choose a programming language and application. Review the code for security vulnerabilities and provide recommendations for secure coding practices. Use tools like static code analyzers or manual code review.

* let's choose the Python programming language and a simple web application developed using the Flask framework. We'll review the code for security vulnerabilities and provide recommendations for secure coding practices. We'll use a combination of manual code review and static code analysis tools like Bandit.



**Security Vulnerabilities:**

Sensitive Information Exposure: The code directly compares the username and password in plain text, which can lead to exposure of sensitive information if intercepted. Storing passwords in plaintext is a significant security risk.

**Weak Password Hashing**: The application is using the outdated and insecure MD5 hashing algorithm to hash passwords. MD5 is vulnerable to collision attacks and should not be used for password hashing.

**Session Management:** While Flask's built-in session management is used, the application does not set a secure SESSION\_COOKIE\_SECURE flag to ensure that session cookies are only sent over HTTPS.

**Cross-Site Scripting (XSS**): The application does not properly sanitize user inputs, leaving it vulnerable to XSS attacks if user inputs are displayed without escaping in the templates.

**Lack of Rate Limiting**: The application does not implement rate limiting on login attempts, making it susceptible to brute force attacks.

**Recommendations for Secure Coding Practices:**

**Use Strong Password Hashing**: Replace MD5 with a more secure hashing algorithm like bcrypt or Argon2 for password storage. These algorithms are specifically designed for secure password hashing and provide better protection against brute force attacks.

**Enable Secure Session Cookies**: Set the SESSION\_COOKIE\_SECURE flag to True in the Flask application configuration to ensure that session cookies are only sent over HTTPS.

**Input Sanitization and Output Encoding**: Implement input sanitization and output encoding to prevent XSS attacks. Use Flask's escape() function to escape user inputs before rendering them in templates.

**Implement Rate Limiting**: Add rate limiting on login attempts to mitigate brute force attacks. You can use Flask extensions like Flask-Limiter to easily implement rate limiting in the application.

**Regular Security Audits**: Perform regular security audits and code reviews using tools like Bandit, as well as manual code review, to identify and address security vulnerabilities in the codebase.

By addressing these vulnerabilities and following secure coding practices, you can significantly enhance the security of the Python Flask application and protect it against common security threats.