**Implements Secure Coding Practices for Task Manager Application**

This document highlights the secure coding practices implemented in the Task Manager Application. These practices ensure that the application is resistant to common security threats and that sensitive data is handled securely.

1. Password Security

The application ensures that user passwords are hashed using the hashlib library before they are stored in the database. This prevents passwords from being stored in plain text, making them more secure in case of a data breach.

Code Example:

import hashlib

def hash\_password(password):

return hashlib.sha256(password.encode()).hexdigest()

Passwords are hashed using the SHA-256 algorithm. This means that the original password is not stored anywhere in the system, making it more difficult for attackers to retrieve user credentials.

1. Input Validation

The application performs input validation to ensure that all user inputs are correctly formatted and meet the necessary requirements before they are processed or stored. This prevents malicious or incorrect data from being entered into the system.

Code Example:

def is\_valid\_date(date\_str):

try:

datetime.datetime.strptime(date\_str, '%m/%d/%y')

return True

except ValueError:

return FalseSecure Error Handling

The application validates that dates are entered in the correct format (MM/DD/YY). If the format is invalid, an error message is shown to the user and the input is rejected.

1. Error Handling

To prevent the exposure of internal system details, the application implements secure error handling practices. Errors are caught and handled in a way that protects sensitive information from being displayed to the user.

Code Example:

try:

# Code that may throw an error

except Exception as e:

print("An error occurred. Please try again.")

Instead of exposing the error details to the user, the system logs the error internally, and the user sees a general error message. This ensures that system details are not exposed in case of an error.

1. Database Security

The application ensures secure interaction with the database by using parameterized queries. This prevents SQL injection attacks, where malicious users could try to inject SQL commands into the application.

Code Example:

def add\_task(task\_title, task\_description):

query = "INSERT INTO tasks (title, description) VALUES (?, ?)"

cursor.execute(query, (task\_title, task\_description))

The use of parameterized queries prevents SQL injection attacks. Instead of directly including user inputs in SQL statements, parameters are used, which safely handle the inputs.

1. Access Control:

The application includes basic access control, where users are required to log in before they can access certain features. This ensures that only authenticated users can create and manage tasks.

Code Example:

def validate\_login(username, password):

hashed\_password = hashlib.sha256(password.encode()).hexdigest()

if username in users and users[username] == hashed\_password:

print("Login successful!")

# Proceed with login

else:

print("Invalid username or password")

The system hashes the user’s password at login and compares it to the hashed password stored in the system. This ensures that passwords are not checked in plain text, further securing user credentials.

By implementing password hashing, input validation, secure error handling, parameterized queries, and access control, the Task Manager Application follows key secure coding practices to protect both the application and its users from common security threats.