Objective:

The objective of this exercise is to implement input validation for a user authentication system to prevent SQL injection attacks. We will use Test-Driven Development (TDD) principles to guide the development process, ensuring that the system behaves as expected and remains secure.

Steps:

1. *Write Test:*
   - Begin by writing test cases that specify the expected behavior of the input validation system.
   - Define test cases for various scenarios, including valid and invalid user inputs, potential SQL injection attacks, and edge cases.
   - Ensure that the test cases cover different types of input data, such as strings, integers, and special characters.
2. *Run Test:*
   - Execute the test cases using a testing framework like pytest.
   - Expect the test cases to fail initially since the input validation implementation does not exist yet.
3. *Write Code:*
   - Implement input validation logic to sanitize and validate user inputs, ensuring they are safe for database operations.
   - Include mechanisms to detect and reject input data that could potentially lead to SQL injection attacks.
   - Utilize techniques such as parameterized queries or ORM libraries to prevent direct execution of user-provided SQL queries.
4. *Run Test Again:*
   - Re-run the test cases to validate the correctness and effectiveness of the input validation implementation.
   - Ensure that the implemented logic successfully prevents SQL injection attacks and handles invalid inputs gracefully.
   - Expect all test cases to pass if the input validation system functions as intended.
5. *Refactor (Optional):*
   - Refactor the input validation code as needed to improve readability, performance, or maintainability.
   - Review and optimize the implementation while ensuring that all test cases continue to pass.
   - Consider additional security measures or best practices to further enhance the system's resilience against potential threats.

By following these steps, you can develop a secure user authentication system that effectively prevents SQL injection attacks and ensures the integrity of your application's data. This approach emphasizes proactive security measures and rigorous testing to mitigate risks associated with user input vulnerabilities.

Now, let's proceed to write the test cases for the input validation system, focusing on security-related scenarios and potential SQL injection attacks.