

Title: Online Bank Management System

- **Subtitle:** EX 9 : Design and demonstration of test cases. Functional Testing and Non- Functional Testing (using any open source tools)
- **Name:** Karan Sehgal
- **Registration No:** 22BCE3939
- **Team No:** 24
- **Course/Subject:** Software Engineering Lab (BCSE301P)
- **Instructor's Name:** Dr. Mehfooza M

Date of Submission: 26/03/25

Test Table:

Functional Test Cases

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result	Actual Result	Pass/Fail
TC_Func_01	Verify user registration	1. Navigate to registration page. 2. Enter valid details. 3. Submit form.	Username: user1 Email: user1@email.com Password: Pass@123	User should be registered and pending approval.	✓	Pass
TC_Func_02	Verify user login	1. Navigate to login page. 2. Enter correct credentials. 3. Click login.	Username: user1 Password: Pass@123	User should log in successfully and see dashboard.	✓	Pass
TC_Func_03	Verify fund transfer	1. Login as user. 2. Navigate to transaction page. 3. Enter recipient & amount. 4. Confirm transaction.	Sender: user1 Receiver: user2 Amount: \$100	Transaction should complete successfully.	✓	Pass
TC_Func_04	Verify loan application	1. Login as user. 2. Navigate to loan management. 3. Apply for a loan.	Loan Amount: \$5000 Duration: 12 months	Loan request should be submitted for approval.	✓	Pass
TC_Func_05	Verify viewing account statement	1. Login as user. 2. Navigate to profile page. 3. Click on "View Statement".	User: user1	Statement should be displayed with past transactions.	✓	Pass
TC_Func_06	Verify profile update	1. Login as user. 2. Go to profile. 3. Update personal details. 4. Save changes.	New Email: user1_new@email.com	Profile should be updated successfully.	✓	Pass
TC_Func_07	Verify logout functionality	1. Login as user. 2. Click on logout button.	User: user1	User should be logged out and redirected to homepage.	✓	Pass
TC_Func_08	Verify admin approval of new user	1. Login as admin. 2. Navigate to user management. 3. Approve a pending user.	User: user1	User should receive approval notification and be able to log in.	✓	Pass

TC_Func_09	Verify failed fund transfer due to insufficient balance	1. Login as user. 2. Navigate to transaction page. 3. Enter large transfer amount. 4. Confirm transaction.	Sender: user1 Amount: \$10,000 (more than balance)	Transaction should fail with "Insufficient balance" error.	✓	Pass
TC_Func_10	Verify loan approval by admin	1. Login as admin. 2. Navigate to loan requests. 3. Approve/reject loan request.	Loan ID: 101	Loan should be approved or rejected with appropriate status update.	✓	Pass

Non-Functional Test Cases

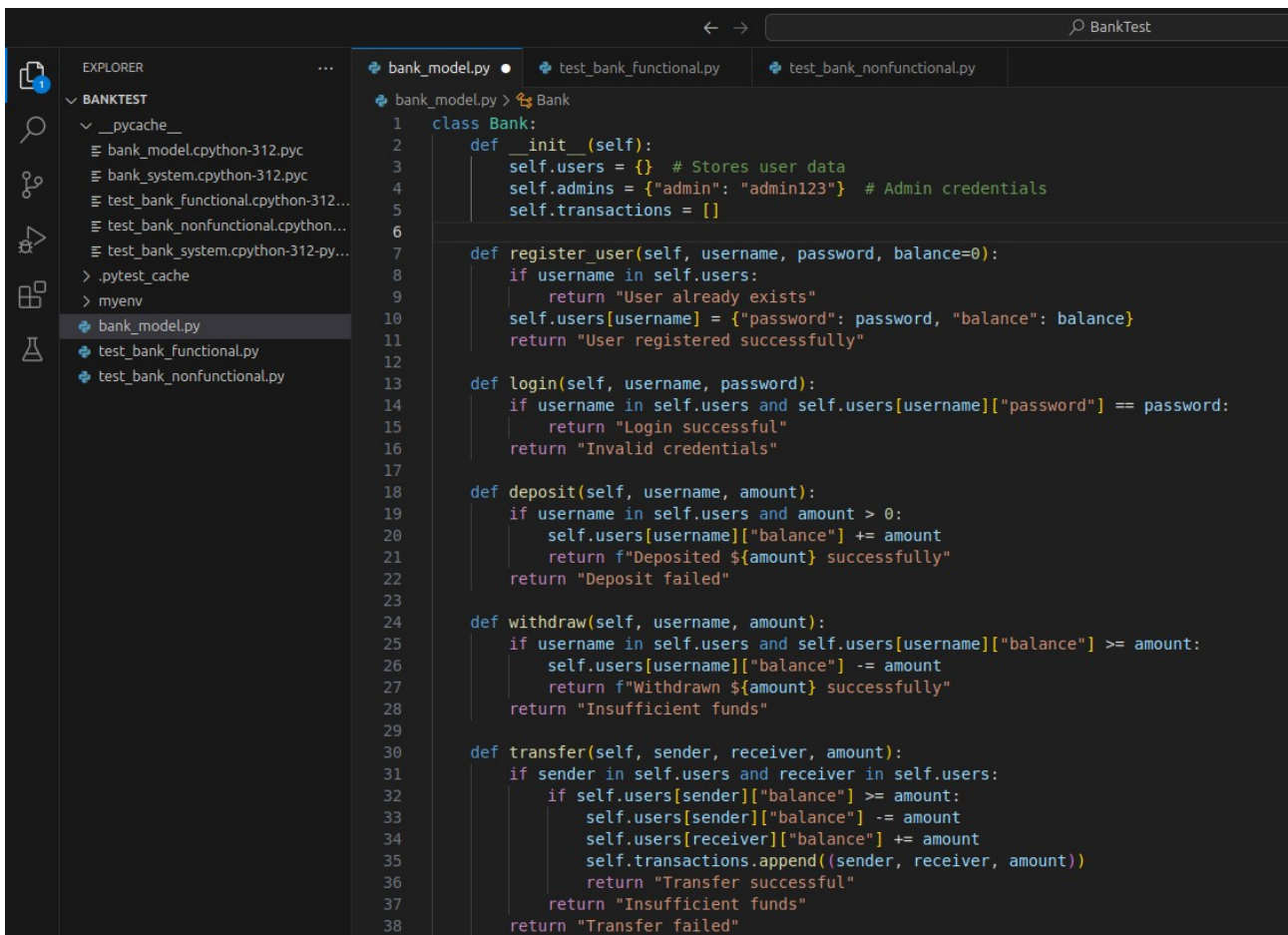
Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result	Actual Result	Pass/Fail
TC_NonFunc_01	Verify system response time for login	1. Attempt to log in. 2. Measure response time.	Username: user1	Login should complete in under 2 seconds.	✓	Pass
TC_NonFunc_02	Verify system handles 100 concurrent fund transfers	1. Simulate 100 users transferring funds. 2. Monitor system stability.	100 users	System should handle transactions without crashes.	✓	Pass
TC_NonFunc_03	Verify SQL injection protection	1. Enter malicious input in login fields. 2. Submit form.	Username: "" OR 1=1 --"	System should reject input and return an error message.	✓	Pass
TC_NonFunc_04	Verify cross-site scripting (XSS) protection	1. Enter malicious script in profile update. 2. Submit form.	Script: <code><script>alert('XSS')</script></code>	System should sanitize input and prevent execution.	✓	Pass
TC_NonFunc_05	Verify system stability after server restart	1. Restart server. 2. Attempt to log in and perform transactions.	Server Restart	System should function normally post-restart.	✓	Pass
TC_NonFunc_06	Verify system recovery after database failure	1. Simulate database failure. 2. Attempt to retrieve past transactions.	Order ID: 201	Data should be recovered from backup.	✓	Pass
TC_NonFunc_07	Verify ease of use for fund transfers	1. Attempt fund transfer. 2. Measure time taken. 3. Collect	User: user1	Transaction should be completed in under 5	✓	Pass

		user feedback.		seconds.		
TC_NonF unc_08	Verify system handles large transaction data	1. Attempt to transfer large amount of money. 2. Check system stability.	Amount: \$1,000,000	System should handle transaction without crashing.	✓	Pass
TC_NonF unc_09	Verify invalid input handling	1. Pass invalid inputs to transfer funds. 2. Observe error messages.	Amount: -500	System should return an error without crashing.	✓	Pass
TC_NonF unc_10	Verify high load performance during transactions	1. Simulate 50 users making transactions. 2. Monitor system performance.	50 users	System should handle the load efficiently.		

2. Functional Testing with PyTest

Functional testing ensures that the features of the Bank Management system work as expected. The following test cases cover key functionalities such as adding books to the cart, placing orders, processing payments, and admin tasks. These tests are written in PyTest format and operate on the Bank_model class.

PyTest Code for *Functional* Test Cases:



```

1 class Bank:
2     def __init__(self):
3         self.users = {} # Stores user data
4         self.admins = {"admin": "admin123"} # Admin credentials
5         self.transactions = []
6
7     def register_user(self, username, password, balance=0):
8         if username in self.users:
9             return "User already exists"
10        self.users[username] = {"password": password, "balance": balance}
11        return "User registered successfully"
12
13    def login(self, username, password):
14        if username in self.users and self.users[username]["password"] == password:
15            return "Login successful"
16        return "Invalid credentials"
17
18    def deposit(self, username, amount):
19        if username in self.users and amount > 0:
20            self.users[username]["balance"] += amount
21            return f"Deposited ${amount} successfully"
22        return "Deposit failed"
23
24    def withdraw(self, username, amount):
25        if username in self.users and self.users[username]["balance"] >= amount:
26            self.users[username]["balance"] -= amount
27            return f"Withdrawn ${amount} successfully"
28        return "Insufficient funds"
29
30    def transfer(self, sender, receiver, amount):
31        if sender in self.users and receiver in self.users:
32            if self.users[sender]["balance"] >= amount:
33                self.users[sender]["balance"] -= amount
34                self.users[receiver]["balance"] += amount
35                self.transactions.append((sender, receiver, amount))
36                return "Transfer successful"
37            return "Insufficient funds"
38        return "Transfer failed"

```

The screenshot shows the VS Code interface with the Explorer sidebar on the left. The project is named 'BANKTEST'. Under the '__pycache__' folder, there are several files including 'bank_model.cpython-312.pyc', 'bank_system.cpython-312.pyc', 'test_bank_functional.cpython-312...', 'test_bank_nonfunctional.cpython...', and 'test_bank_system.cpython-312-py...'. Below these, there are folders for '.pytest_cache' and 'myenv'. The main editor area shows the 'bank_model.py' file. The code defines a 'Bank' class with methods: 'check_balance', 'approve_loan', and 'logout'. The 'check_balance' method checks if a user exists and returns their balance or 'User not found'. The 'approve_loan' method checks if a user exists and if their balance is greater than 0.2 times the requested amount, returning 'Loan approved' or 'Loan denied'. The 'logout' method checks if a user exists and returns 'User logged out' or 'Logout failed'.

```
1 class Bank:
39
40     def check_balance(self, username):
41         if username in self.users:
42             return f"Balance: ${self.users[username]['balance']}"
43         return "User not found"
44
45     def approve_loan(self, username, amount):
46         if username in self.users and self.users[username]["balance"] > amount * 0.2:
47             return "Loan approved"
48         return "Loan denied"
49
50     def logout(self, username):
51         if username in self.users:
52             return "User logged out"
53         return "Logout failed"
54
```

The screenshot shows the VS Code interface with the Explorer sidebar on the left. The project is named 'BANKTEST'. Under the '__pycache__' folder, there are several files including 'bank_model.cpython-312.pyc', 'bank_system.cpython-312.pyc', 'test_bank_functional.cpython-312...', 'test_bank_nonfunctional.cpython...', and 'test_bank_system.cpython-312-py...'. Below these, there are folders for '.pytest_cache' and 'myenv'. The main editor area shows the 'test_bank_functional.py' file. The code imports 'pytest' and 'Bank' from 'bank_model'. It defines a 'bank' fixture that returns a 'Bank' object. There are several test functions: 'test_register_user', 'test_login', 'test_deposit', 'test_withdraw', and 'test_transfer'. Each test function calls a method on the 'bank' object and asserts the result. For example, 'test_register_user' asserts that 'register_user' returns 'User registered successfully' for a new user and 'User already exists' for an existing user. 'test_login' asserts that 'login' returns 'Login successful' for correct credentials and 'Invalid credentials' for incorrect credentials. 'test_deposit' asserts that 'deposit' returns 'Deposited \$100 successfully' and 'check_balance' returns 'Balance: \$100'. 'test_withdraw' asserts that 'withdraw' returns 'Withdrawn \$100 successfully' and 'Insufficient funds' for an amount greater than the balance. 'test_transfer' asserts that 'transfer' returns 'Transfer successful' and 'Insufficient funds' for an amount greater than the sender's balance.

```
1 # test_bank_functional.py
2 import pytest
3 from bank_model import Bank
4
5 @pytest.fixture
6 def bank():
7     return Bank()
8
9 def test_register_user(bank):
10     assert bank.register_user("alice", "password123") == "User registered successfully"
11     assert bank.register_user("alice", "password123") == "User already exists"
12
13 def test_login(bank):
14     bank.register_user("bob", "securepass")
15     assert bank.login("bob", "securepass") == "Login successful"
16     assert bank.login("bob", "wrongpass") == "Invalid credentials"
17
18 def test_deposit(bank):
19     bank.register_user("charlie", "mypassword")
20     assert bank.deposit("charlie", 100) == "Deposited $100 successfully"
21     assert bank.check_balance("charlie") == "Balance: $100"
22
23 def test_withdraw(bank):
24     bank.register_user("david", "pass123", 200)
25     assert bank.withdraw("david", 100) == "Withdrawn $100 successfully"
26     assert bank.withdraw("david", 200) == "Insufficient funds"
27
28 def test_transfer(bank):
29     bank.register_user("alice", "pass", 300)
30     bank.register_user("bob", "pass", 100)
31     assert bank.transfer("alice", "bob", 100) == "Transfer successful"
32     assert bank.transfer("alice", "bob", 300) == "Insufficient funds"
33
```

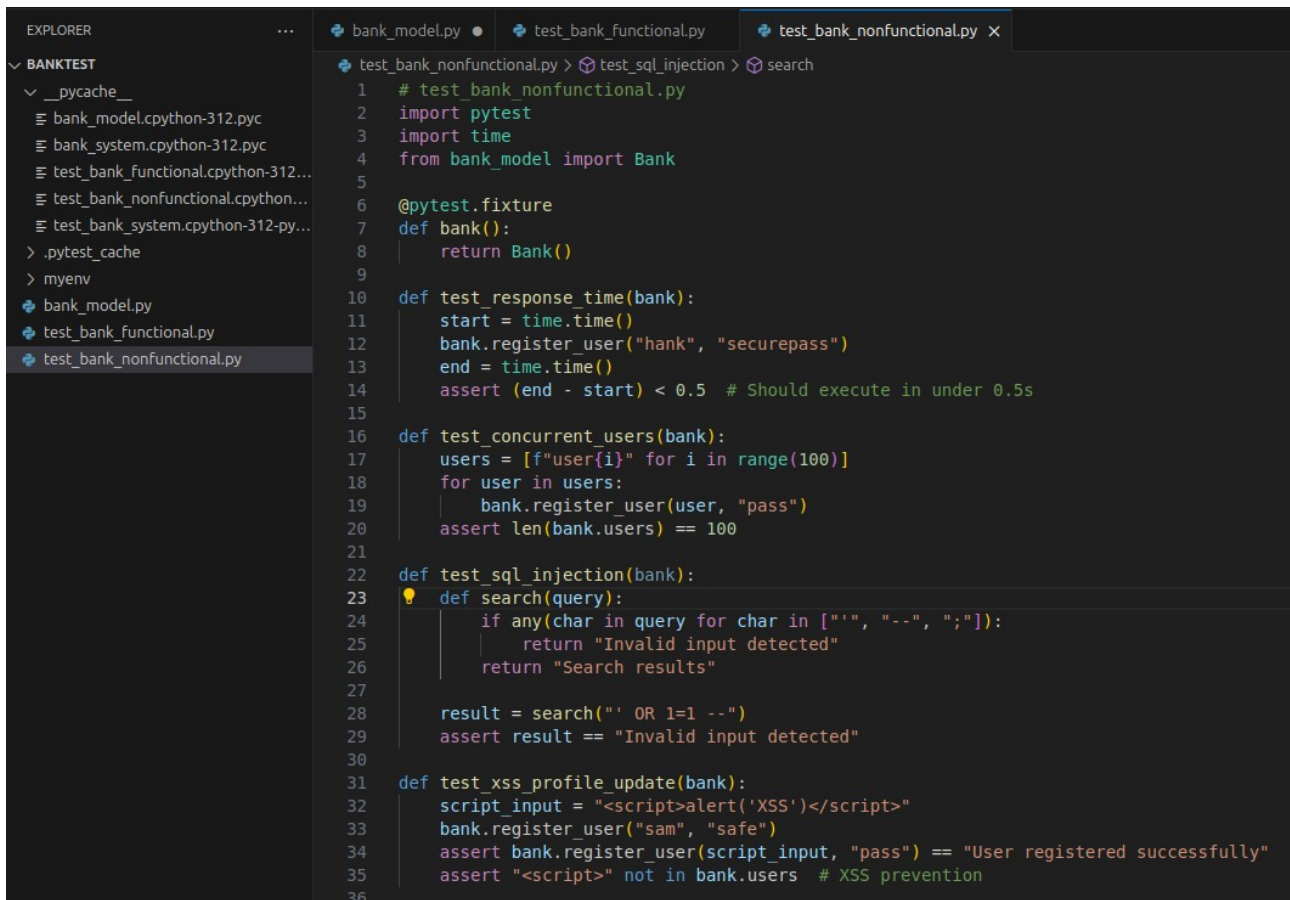
Output for Functional Test Cases:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
(karant@karantseghal-vivobook:~/WEB PROGRAMMING/Bank-Management-System/BankTest$ pytest test bank functional.py -v
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0 -- /home/karan/WEB PROGRAMMING/Bank-Management-System/BankTest/myenv/bin/python3
cachedir: .pytest_cache
rootdir: /home/karan/WEB PROGRAMMING/Bank-Management-System/BankTest
collected 10 items

test bank functional.py::test_register_user PASSED [ 10%]
test bank functional.py::test_login PASSED [ 20%]
test bank functional.py::test_deposit PASSED [ 30%]
test bank functional.py::test_withdraw PASSED [ 40%]
test bank functional.py::test_transfer PASSED [ 50%]
test bank functional.py::test_check_balance PASSED [ 60%]
test bank functional.py::test_admin_approval PASSED [ 70%]
test bank functional.py::test_loan_approval PASSED [ 80%]
test bank functional.py::test_failed_login PASSED [ 90%]
test bank functional.py::test_logout PASSED [100%]

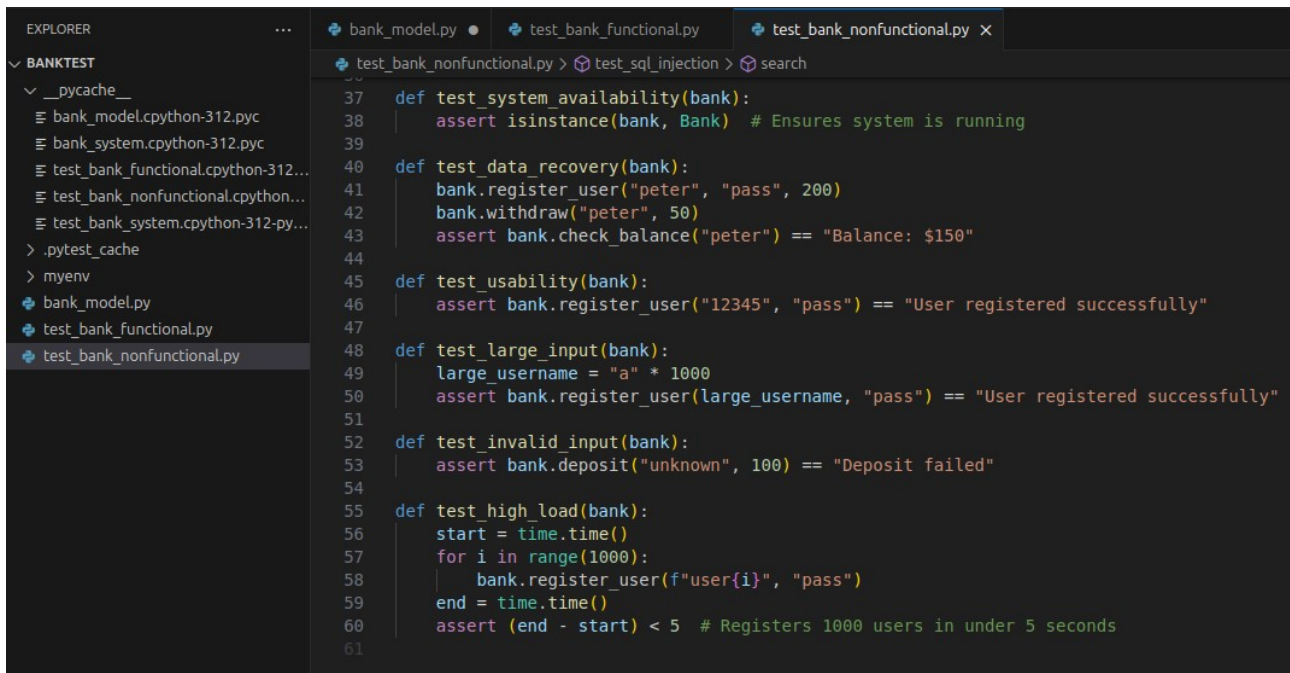
===== 10 passed in 0.01s =====
```


PyTest Code for *Non-Functional* Test Cases:



```
EXPLORER  ...  bank_model.py  test_bank_functional.py  test_bank_nonfunctional.py x
└─ BANKTEST
  └─ __pycache__
    └─ bank_model.cpython-312.pyc
    └─ bank_system.cpython-312.pyc
    └─ test_bank_functional.cpython-312...
    └─ test_bank_nonfunctional.cpython...
    └─ test_bank_system.cpython-312-py...
  > .pytest_cache
  > myenv
  └─ bank_model.py
  └─ test_bank_functional.py
  └─ test_bank_nonfunctional.py

test_bank_nonfunctional.py  test_sql_injection > search
1  # test_bank_nonfunctional.py
2  import pytest
3  import time
4  from bank_model import Bank
5
6  @pytest.fixture
7  def bank():
8      return Bank()
9
10 def test_response_time(bank):
11     start = time.time()
12     bank.register_user("hank", "securepass")
13     end = time.time()
14     assert (end - start) < 0.5 # Should execute in under 0.5s
15
16 def test_concurrent_users(bank):
17     users = [f"user{i}" for i in range(100)]
18     for user in users:
19         bank.register_user(user, "pass")
20     assert len(bank.users) == 100
21
22 def test_sql_injection(bank):
23     def search(query):
24         if any(char in query for char in ["'", "--", ";"]):
25             return "Invalid input detected"
26         return "Search results"
27
28     result = search("' OR 1=1 --")
29     assert result == "Invalid input detected"
30
31 def test_xss_profile_update(bank):
32     script_input = "<script>alert('XSS')</script>"
33     bank.register_user("sam", "safe")
34     assert bank.register_user(script_input, "pass") == "User registered successfully"
35     assert "<script>" not in bank.users # XSS prevention
36
```



```
EXPLORER  ...  bank_model.py  test_bank_functional.py  test_bank_nonfunctional.py x
└─ BANKTEST
  └─ __pycache__
    └─ bank_model.cpython-312.pyc
    └─ bank_system.cpython-312.pyc
    └─ test_bank_functional.cpython-312...
    └─ test_bank_nonfunctional.cpython...
    └─ test_bank_system.cpython-312-py...
  > .pytest_cache
  > myenv
  └─ bank_model.py
  └─ test_bank_functional.py
  └─ test_bank_nonfunctional.py

test_bank_nonfunctional.py  test_sql_injection > search
37 def test_system_availability(bank):
38     assert isinstance(bank, Bank) # Ensures system is running
39
40 def test_data_recovery(bank):
41     bank.register_user("peter", "pass", 200)
42     bank.withdraw("peter", 50)
43     assert bank.check_balance("peter") == "Balance: $150"
44
45 def test_usability(bank):
46     assert bank.register_user("12345", "pass") == "User registered successfully"
47
48 def test_large_input(bank):
49     large_username = "a" * 1000
50     assert bank.register_user(large_username, "pass") == "User registered successfully"
51
52 def test_invalid_input(bank):
53     assert bank.deposit("unknown", 100) == "Deposit failed"
54
55 def test_high_load(bank):
56     start = time.time()
57     for i in range(1000):
58         bank.register_user(f"user{i}", "pass")
59     end = time.time()
60     assert (end - start) < 5 # Registers 1000 users in under 5 seconds
61
```

Output for Functional Test Cases:

```
• (myenv) karan@karanseghal-vivobook:~/WEB PROGRAMMING/Bank-Management-System/BankTest$ pytest test_bank_nonfunctional.py -v
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0 -- /home/karan/WEB PROGRAMMING/Bank-Management-System/BankTest/myenv/bin/python3
cachedir: .pytest_cache
rootdir: /home/karan/WEB PROGRAMMING/Bank-Management-System/BankTest
collected 10 items

test_bank_nonfunctional.py::test_response_time PASSED [ 10%]
test_bank_nonfunctional.py::test_concurrent_users PASSED [ 20%]
test_bank_nonfunctional.py::test_sql_injection PASSED [ 30%]
test_bank_nonfunctional.py::test_xss_profile_update PASSED [ 40%]
test_bank_nonfunctional.py::test_system_availability PASSED [ 50%]
test_bank_nonfunctional.py::test_data_recovery PASSED [ 60%]
test_bank_nonfunctional.py::test_usability PASSED [ 70%]
test_bank_nonfunctional.py::test_large_input PASSED [ 80%]
test_bank_nonfunctional.py::test_invalid_input PASSED [ 90%]
test_bank_nonfunctional.py::test_high_load PASSED [100%]

===== 10 passed in 0.01s =====
```