Functional Test Cases

Scenario 1: User Creation(create_user Endpoint)

1. **Test Case 1: Valid User Creation**
    - **Test Case ID:** TC001
    - **Description:** Verify that a new user account can be successfully created.
    - **Test Steps:**
        1. Send a POST request to the create_user endpoint with a valid username.
        2. Verify that the response status is successful.
2. **Test Case 2: Duplicate User Creation**
    - **Test Case ID:** TC002
    - **Description:** Verify that an appropriate error message is displayed after a duplicate username creation request.
    - **Test Steps:**
        1. Create a user with a specific username.
        2. Attempt to create another user with the same username.
        3. Verify that the response status indicates failure and provides an appropriate error message.
3. **Test Case 3: Empty Username**
    - **Test Case ID:** TC003
    - **Description:** Verify that an appropriate error message is displayed when an empty username is provided.
    - **Test Steps:**
        1. Send a POST request to the create_user endpoint with an empty username.
        2. Verify that the response status indicates failure and provides an appropriate error message.
4. **Test Case 4: Username with Special Characters**
    - **Test Case ID:** TC004
    - **Description:** Verify that an appropriate error message is displayed when the username contains special characters.
    - **Test Steps:**
        1. Attempt to create a user with a username containing special characters (e.g., @, #, $).
        2. Verify that the response status indicates failure and provides an appropriate error message.
5. **Test Case 5: Long Username (Boundary Testing)**
    - **Test Case ID:** TC005
    - **Description:** Verify that an appropriate error message is displayed when the username exceeds the maximum character limit.
    - **Test Steps:**
        1. Create a user with a username exceeding the maximum character limit.
        2. Verify that the response status indicates failure and provides an appropriate error message.

6. **Test Case 6: Concurrent User Creations**
    - **Test Case ID:** TC006
    - **Description:** Verify that multiple users can be created simultaneously without data corruption or inconsistencies.
    - **Test Steps:**
        1. Simultaneously send multiple POST requests to the create_user endpoint with different usernames.
        2. Verify that all users are successfully created without data corruption or inconsistencies.
7. **Test Case 7: Maximum Number of Users**
    - **Test Case ID:** TC007
    - **Description:** Verify that the system can handle the maximum allowed number of user creations.
    - **Test Steps:**
        1. Attempt to create the maximum allowed number of users.
        2. Verify that all users are successfully created without any system errors.

Scenario 2: Depositing Money(deposit Endpoint)

8. **Test Case 8: Valid Deposit**

- **Test Case ID:** TC008
- **Description:** Verify that a user can successfully deposit money into their account.
- **Test Steps:**

    1. Send a POST request to the deposit endpoint with a valid username and a positive amount to deposit.
    2. Verify that the response status is successful.
    3. Check if the deposit is reflected in the user's account balance correctly.

9. **Test Case 9: Invalid User for Deposit**
    - **Test Case ID:** TC009
    - **Description:** Verify that an appropriate error message is displayed when depositing to a non-existent user.
    - **Test Steps:**
        1. Send a POST request to the deposit endpoint with a non-existing username and a positive amount to deposit.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the user does not exist.

10. **Test Case 10: Negative Deposit Amount**
    - **Test Case ID:** TC010
    - **Description:** Verify that an appropriate error message is displayed when a negative amount is deposited.
    - **Test Steps:**
        1. Send a POST request to the deposit endpoint with a valid username and a negative amount to deposit.
        2. Verify that the response status indicates failure (4xx or 5xx error code).
        3. Ensure that the error message specifies that the deposit amount is invalid.
11. **Test Case 11: Zero Deposit Amount**
    - **Test Case ID:** TC011
    - **Description:** Verify that an appropriate error message is displayed when a zero amount is deposited.
    - **Test Steps:**
        1. Send a POST request to the deposit endpoint with a valid username and zero amount to deposit.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the deposit amount must be greater than zero.
12. **Test Case 12: Large Deposit Amount (Boundary Testing)**
    - **Test Case ID:** TC012
    - **Description:** Verify that an appropriate error message is displayed when the deposit amount exceeds the system's maximum limit.
    - **Test Steps:**
        1. Send a POST request to the deposit endpoint with a valid username and a deposit amount exceeding the system's maximum limit.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the deposit amount exceeds the maximum limit.
13. **Test Case 13: Concurrent Deposits**
    - **Test Case ID:** TC013
    - **Description:** Verify that multiple deposits can be made simultaneously without data corruption or inconsistencies.
    - **Test Steps:**
        1. Simultaneously send multiple POST requests to the deposit endpoint with the same or different usernames and valid deposit amounts.
        2. Verify that all deposits are processed successfully without data corruption or inconsistencies.

Scenario 3: Withdrawing Money(withdraw Endpoint)

14. **Test Case 14: Valid Withdrawal -**
    - **Test Case ID: TC014**
    - **Description:** Verify that a user can successfully withdraw money from their account.
    - **Test Steps:**
      1. Send a POST request to the withdraw endpoint with a valid username and a positive amount to withdraw.
      2. Verify that the response status is successful.
      3. Check if the withdrawal is reflected in the user's account balance correctly.

15. **Test Case 15: Invalid User for Withdrawal**
    - **Test Case ID:** TC015
    - **Description:** Verify that an appropriate error message is displayed when withdrawing from a non-existent user.
    - **Test Steps:**
      1. Send a POST request to the withdraw endpoint with a non-existing username and a positive amount to withdraw.
      2. Verify that the response status indicates failure.
      3. Ensure that the error message specifies that the user does not exist.
16. **Test Case 16: Insufficient Balance for Withdrawal**
    - **Test Case ID:** TC016
    - **Description:** Verify that an appropriate error message is displayed when the withdrawal amount exceeds the user's balance.
    - **Test Steps:**
      1. Send a POST request to the withdraw endpoint with a valid username and an amount exceeding the user's balance to withdraw.
      2. Verify that the response status indicates failure.
      3. Ensure that the error message specifies that the withdrawal amount exceeds the available balance.
17. **Test Case 17: Negative Withdrawal Amount**
    - **Test Case ID:** TC017
    - **Description:** Verify that an appropriate error message is displayed when a negative amount is withdrawn.
    - **Test Steps:**
      1. Send a POST request to the withdraw endpoint with a valid username and a negative amount to withdraw.
      2. Verify that the response status indicates failure.
      3. Ensure that the error message specifies that the withdrawal amount is invalid.

18. **Test Case 18: Zero Withdrawal Amount**
    - **Test Case ID:** TC018
    - **Description:** Verify that an appropriate error message is displayed when a zero amount is withdrawn.
    - **Test Steps:**
        1. Send a POST request to the withdraw endpoint with a valid username and zero amount to withdraw.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the withdrawal amount must be greater than zero.
19. **Test Case 19: Large Withdrawal Amount (Boundary Testing)**
    - **Test Case ID:** TC019
    - **Description:** Verify that an appropriate error message is displayed when the withdrawal amount exceeds the system's maximum limit.
    - **Test Steps:**
        1. Send a POST request to the withdraw endpoint with a valid username and a withdrawal amount exceeding the system's maximum limit.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the withdrawal amount exceeds the maximum limit.
20. **Test Case 20: Concurrent Withdrawals**
    - **Test Case ID:** TC020
    - **Description:** Verify that multiple withdrawals can be made simultaneously without data corruption or inconsistencies.
    - **Test Steps:**
        1. Simultaneously send multiple POST requests to the withdraw endpoint with the same or different usernames and valid withdrawal amounts.
        2. Verify that all withdrawals are processed successfully without data corruption or inconsistencies.

Scenario 4: Balance Inquiry(get_balance Endpoint)

21. **Test Case 21: Valid Balance Inquiry**

    Test Case ID: TC021
    Description: Verify that a user can successfully inquire about their account balance.
    Test Steps:
    1. Send a POST request to the get_balance endpoint with a valid username.
    2. Verify that the response status is successful.
    3. Check if the returned balance matches the actual balance of the user.

22. **Test Case 22: Non-existing User for Balance Inquiry**
    - **Test Case ID:** TC022
    - **Description:** Verify that an appropriate error message is displayed when inquiring about a non-existent user's balance.
    - **Test Steps:**
        1. Send a POST request to the get_balance endpoint with a non-existing username.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the user does not exist.
23. **Test Case 23: Concurrent Balance Inquiries**
    - **Test Case ID:** TC023
    - **Description:** Verify that multiple balance inquiries can be made simultaneously without data corruption or inconsistencies.
    - **Test Steps:**
        1. Simultaneously send multiple POST requests to the get_balance endpoint with the same or different usernames.
        2. Verify that all balance inquiries return accurate results without data corruption or inconsistencies.

Scenario 5: Money Transfer(send Endpoint)

24. **Test Case 24: Valid Money Transfer -**
    - **Test Case ID:** TC024
    - **Description:** Verify that a user can successfully transfer money to another user's account.
    - **Test Steps:**
      1. Send a POST request to the send endpoint with valid sender username, receiver username, and a positive amount to transfer.
      2. Verify that the response status is successful.
      3. Check if the transfer is reflected in the sender's and receiver's account balances correctly.
25. **Test Case 25: Insufficient Balance for Transfer**
    - **Test Case ID:** TC025
    - **Description:** Verify that an appropriate error message is displayed when the transfer amount exceeds the sender's balance.
    - **Test Steps:**
        1. Send a POST request to the send endpoint with a valid sender username, receiver username, and an amount exceeding the sender's balance to transfer.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the sender has insufficient funds for the transfer.

26. **Test Case 26: Transfer to Non-existing Receiver**
    - **Test Case ID:** TC026
    - **Description:** Verify that an appropriate error message is displayed when transferring money to a non-existent user.
    - **Test Steps:**
        1. Send a POST request to the send endpoint with a valid sender username, non-existing receiver username, and a positive amount to transfer.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the receiver does not exist.
27. **Test Case 27: Negative Transfer Amount**
    - **Test Case ID:** TC027
    - **Description:** Verify that an appropriate error message is displayed when a negative transfer amount is specified.
    - **Test Steps:**
        1. Send a POST request to the send endpoint with valid sender username, receiver username, and a negative amount to transfer.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the transfer amount is invalid.
28. **Test Case 28: Zero Transfer Amount**
    - **Test Case ID:** TC028
    - **Description:** Verify that an appropriate error message is displayed when a zero transfer amount is specified.
    - **Test Steps:**
        1. Send a POST request to the send endpoint with valid sender username, receiver username, and zero amount to transfer.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the transfer amount must be greater than zero.
29. **Test Case 29: Large Transfer Amount (Boundary Testing)**
    - **Test Case ID:** TC029
    - **Description:** Verify that an appropriate error message is displayed when the transfer amount exceeds the system's maximum limit.
    - **Test Steps:**
        1. Send a POST request to the send endpoint with valid sender username, receiver username, and a transfer amount exceeding the system's maximum limit.
        2. Verify that the response status indicates failure.
        3. Ensure that the error message specifies that the transfer amount exceeds the maximum limit.

30. **Test Case 30: Concurrent Transfers**
    - **Test Case ID:** TC030
    - **Description:** Verify that multiple transfers can be made simultaneously without data corruption or inconsistencies.
    - **Test Steps:**
        1. Simultaneously send multiple POST requests to the send endpoint with different sender usernames, receiver usernames, and valid transfer amounts.
        2. Verify that all transfers are processed successfully without data corruption or inconsistencies.
        3. Check if the balances of both sender and receiver are updated correctly after each transfer.
31. **Test Case 31: Long Sequence of Deposits and Withdrawals**
    - **Test Case ID:** TC031
    - **Description:** Verify that the system maintains consistency and correctness after a long sequence of deposit and withdrawal operations.
    - **Test Steps:**
        1. Perform a series of deposit and withdrawal operations on multiple user accounts over an extended period.
        2. Verify that all transactions are executed successfully without errors.
        3. Check the consistency of account balances after each transaction.
32. **Test Case 32: Varying Transfer Frequencies**
    - **Test Case ID: TC032**
    - **Description:** Send multiple transfer requests with varying time intervals between each transfer.
    - **Test Steps:**

1. Send multiple POST requests to the send endpoint with different time intervals between each transfer.
2. Monitor the system's performance and stability under different transfer frequencies.
3. Verify that the system can handle both high and low transfer frequencies without degradation in performance.

33. **Test Case 33: Randomized User Interactions**
    - **Test Case ID: TC033**
    - **Description:** Randomly select pairs of users and perform deposit, withdrawal, and transfer operations between them.
    - **Test Steps:**

1. Randomly select pairs of users.
2. Perform deposit, withdrawal, and transfer operations between the selected users.
3. Ensure that the system remains stable and responsive despite random user interactions.
4. Validate the accuracy of account balances after each interaction.

Non-Functional Test Cases

Scenario 6: Stress and Performance Testing

34. **Test Case 34: Stress Testing with Concurrent Operations**

- **Test Case ID:** TC034
- **Description:** Simulate a high load scenario by executing a large number of concurrent user creation operations.
- **Test Steps:**
    1. Generate a load with an arrival rate of 5 requests per second for 60 seconds.
    2. Verify that the system can handle the load without crashing or experiencing significant performance degradation.

35. **Test Case 35: Network Latency Testing**

- **Test Case ID:** TC035
- **Description:** Introduce artificial network latency between the client and server using network emulation tools.
- **Test Steps:**
    1. Introduce artificial network latency between the client and server.
    2. Send requests to the ExBanking service endpoints and observe the response times under different latency conditions.
    3. Ensure that the system maintains acceptable response times even under high network latency.

36. **Test Case 36: Test Data Integrity During System Failures**

- **Test Case ID:** TC036
- **Description:** Simulate system failures such as server crashes or network outages during ongoing transactions.
- **Test Steps:**
    1. Simulate system failures such as server crashes or network outages during ongoing transactions.
    2. Verify that the system can recover gracefully from failures without data loss or corruption.
    3. Perform data integrity checks to ensure that all transactions are accurately recorded and reflected in account balances upon recovery.