Functional and Non-Functional Test Cases for a Banking Service.

Table Of Contents:

Functional Test Cases. 2

1. Create User: 2

ExBank-FT-1 2

ExBank-FT-2 2

ExBank-FT-3 3

2. Deposit: 3

ExBank-FT-4 3

ExBank-FT-5 4

ExBank-FT-6 4

ExBank-FT-7 5

3. Withdraw: 6

ExBank-FT-8 6

ExBank-FT-9 6

ExBank-FT-10 7

ExBank-FT-11 7

4. Get Balance: 8

ExBank-FT-12 8

ExBank-FT-13 8

5. Send: 9

ExBank-FT-14 9

ExBank-FT-15 9

ExBank-FT-16 10

ExBank-FT-17 11

ExBank-FT-18 11

ExBank-FT-19 12

Non-Functional test cases: 13

1. Performance: 13

ExBank-NFT-1 13

ExBank-NFT-2 13

ExBank-NFT-3 14

ExBank-NFT-4 14

2. Scalability: 14

ExBank-NFT-5 14

3. Reliability: .... **Tõrge! Järjehoidjat pole määratletud.**
ExBank-NFT-6 .... **Tõrge! Järjehoidjat pole määratletud.**

Functional Test Cases.

baseURL: http://localhost:8080

1. Create User:

Test Case ID:	ExBank-FT-1
Title:	Create a user with a unique username.
Description:	Verify that you can create a new user with a unique username.
Precondition:	The local server is running and has no user with the username “Sample123”

Test data:	{
	“username” : “Sample123”,
	“initial_balance”: 300
	}

Test steps:	1. Send a POST request to the **{baseURL} /api/create_user** with data in body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send **GET** request to the **{baseURL}/api/get_user?username=Sample123**. 5. Verify that you get a response with data.
Expected Result:	The API returns status code 201. Created. The response message. User created successfully. The response code 200 OK The user with username “Sample123” exists on the database.

Test Case ID:	ExBank-FT-2
Title:	Try to create a user with a username that already exists.
Description:	Verify that the system does not allow creating a user with a username already existing in the database.
Precondition:	The local server is running. A user with the username “Sample123” already exists.

Test data:	{
	“username” : “Sample123”
	}

Test steps:	1. Send a POST request to the **{baseURL} /api/create_user** with data in body from test data. 2. Verify the response status code. 3. Verify the response message.
Expected Result:	Response status code 400. Bad Request The response message. The user already exists.

Test Case ID:	ExBank-FT-3
Title:	Create a user with a valid but uncommon username(with special characters).
Description:	The local server is running. Verify that the system allows the creation of a username with special characters.
Precondition:	The local server is running and has no user with the username with special characters.

Test data:	{
	“username”: “user@456!”
	}

Test steps:	1. Send a POST request to the **{baseURL} /api/create_user** with data in body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send **GET** request to the **{baseURL}/api/get_user?username=user@456!**. 5. Verify that you get a response with data.
Expected Result:	Response status code 201. Created Response message. User Created successfully. Response status code 200. OK

2. Deposit:

Test Case ID:	ExBank-FT-4
Title:	Deposit a valid amount to the user's account.
Description:	Verify that the system allows depositing a valid amount to a user's account.
Precondition:	The local server is running. The user already exists in the system.

Test data:	{
	“username”: “Sample123”,
	“amount”: 100
	}

Test steps:	1. Send POST request to **{baseURL}/api/deposit** with data in the body from test data.

	2. Verify the response status code. 3. Verify the response message. 4. Verify that the balance is updated.
Expected Result:	Response status code 200 OK. Response message Deposit successful. Users balance reflects the deposited amount.

Test Case ID:	ExBank-FT-5
Title:	Deposit a zero amount and check the balance.
Description:	Verify that the system handles depositing a zero amount correctly and ensures the balance remains unchanged.
Precondition:	The local Server is running. Test ExBank FT-1 is executed. Balance 0.

Test data:	{
	“username”: “Sample123”,
	“amount” : 0
	}

Test steps:	1. Send POST request to **{baseURL}/api/deposit** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send **GET** request to **{baseURL}/api/get_balance?username=Sample123** 5. Verify that the balance is the same as before the deposit.
Expected Result:	Response status code 400. Bad Request. Response message Deposit amount should be positive. The user's balance is not changed

Test Case ID:	ExBank-FT-6
Title:	Deposit a negative amount and ensure it fails.
Description:	Verify that the system handles depositing a negative amount correctly and ensures that the deposit operation fails.
Precondition:	The local Server is running. Test ExBank FT-1 is executed.

Test data:	{
	“username”: “Sample123”,
	“amount” : -80
	}

Test steps:	1. Send POST request to **{baseURL}/api/deposit** with data in the body from test data.

	2. Verify the response status code. 3. Verify the response message. 4. Send **GET** request to **{baseURL}/api/get_balance?username=Sample123** 5. Verify that the balance is the same as before the deposit.
Expected Result:	Response status code 400. Bad Request Response message Deposit amount should be positive. The user's balance is not changed.

Test Case ID:	ExBank-FT-7
Title:	Deposit a very large amount
Description:	Verify that the system can handle depositing a very large amount without any errors.
Precondition:	The local server is running. The user already exists in the system with a known balance.

Test data:	{
	“username”: “Sample123”,
	“amount” : 1 000 000 000
	}

Test steps:	1. Send POST request to **{baseURL}/api/deposit** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send **GET** request to **{baseURL}/api/get_balance?username=Sample123** 5. Verify that the balance is correctly updated.
Expected Result:	Response status code 200. OK The response message is Deposit successful. Users balance is updated correctly.

3. Withdraw:

Test Case ID:	ExBank-FT-8
Title:	Withdraw a valid amount from the user's account.
Description:	Verify that the system correctly handles withdrawing a valid amount from the user's account and updates the balance correctly.
Precondition:	The local Server is running. Test ExBank FT-1 executed.

Test data:	{
	“username”: “Sample123”,
	“amount_to_withdraw” : 150
	}

Test steps:	1. Send **GET** request **to {baseURL}/api/get_balance?username=Sample123** to check initial balance. 2. Send **POST** request to **{baseURL}/api/withdraw** with data in the body from test data. 3. Verify the response status code. 4. Verify the response message. 5. Verify that the balance is updated.
Expected Result:	Response status 200. OK Response message. Withdrawal successful. The user’s balance is updated correctly according to the amount of withdrawal.

Test Case ID:	ExBank-FT-9
Title:	Withdraw an amount greater than the balance.
Description:	Verify that the user can’t withdraw a greater amount than has on the balance
Precondition:	The local Server is running. The user already exists in the system with a known balance.

Test data:	{
	“username”: “Sample123”,
	“amount_to_withdraw” : “1500”
	}

Test steps:	1. Send **GET** request **to {baseURL}/api/get_balance** to check initial balance. 2. Send **POST** request to **{baseURL}/api/withdraw** with data in the body from test data. 3. Verify the response status code. 4. Verify the response message. 5. Send **GET** request **to {baseURL}/api/get_balance** 6. Verify that the balance is not changed.
Expected Result:	Response code 400. Bad Request Response message. Insufficient funds. Balance is unchanged.

Test Case ID:	ExBank-FT-10
Title:	Withdraw a zero amount.
Description:	Verify that the system correctly handles the scenario when user want to withdraw amount of zero and returns correct error message and balance is not changed.
Precondition:	The local Server is running. The user already exists in the system with a known balance.

Test data:	{
	“username”: “Sample123”,
	“amount_to_withdraw” : 0
	}

Test steps:	1. Send GET request **to {baseURL}/api/get_balance** to check initial balance. 2. Send POST request to **{baseURL}/api/withdraw** with data in the body from test data. 3. Verify the response status code. 4. Verify the response message. 5. Send GET request **to {baseURL}/api/get_balance** 6. Verify that the balance is not changed.
Expected Result:	Response code 400. Bad Request. Response message. Invalid amount Balance is unchanged.

Test Case ID:	ExBank-FT-11
Title:	Withdraw a negative amount and ensure it fails.
Description:	Verify that the system correctly handles the scenario when the user want to withdraw a negative amount and returns the correct error message and the balance is not changed.
Precondition:	The local server is running. The user already exists in the system with a known balance.

Test data:	{
	“username”: “Sample123”,
	“amount_to_withdraw” : -100
	}

Test steps:	1. Send GET request **to {baseURL}/api/get_balance** to check initial balance. 2. Send POST request to **{baseURL}/api/deposit** with data in the body from test data. 3. Verify the response status code. 4. Verify the response message. 5. Send GET request **to {baseURL}/api/get_balance** Verify that the balance is not changed.
Expected Result:	Response code 400. Bad Request Response message. Invalid amount Balance is unchanged.

4. Get Balance:

Test Case ID:	ExBank-FT-12
Title:	Retrieve the balance for an existing user
Description:	Verify that the system correctly retrieves and returns the balance for existing user.
Precondition:	The local server is running. The user already exists in the system with a known balance.
Test data:	Username= Sample123
Test steps:	1. Send **GET** request **to {baseURL}/api/get_balance?username=Sample123** with user name from test data. 2. Verify response code. 3. Verify response message.
Expected Result:	Response code 200. Response message. OK!. Success. Response balance matches the initial balance.

Test Case ID:	ExBank-FT-13
Title:	Retrieve the balance for a non-existing user.
Description:	Verify that the system returns an error message when attempting to retrieve the balance for a non-existing user.
Precondition:	The local Server is running. The user does not exist in the system.
Test data:	-
Test steps:	1.Send **GET** request **to {baseURL}/api/get_balance?username=99999** 2. Verify response code. 3. Verify response message.
Expected Result:	Response code 404 Not Found Response message User not found.

5. Send:

Test Case ID:	ExBank-FT-14
Title:	Send a valid amount from one user to another.
Description:	Verify that system correctly processes a transfer of a valid amount from one user to another and their balances are updated correctly.
Precondition:	The local Server is running. Both users exist with a known and enough balance for transfer.

Test data:	{
	“from” : “Sample123”,
	“to”: “user@456!”,
	“transfer_amount” : 100
	}

Test steps:	1. Send POST request to **{baseURL}/api/send** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send **GET** request **to {baseURL}/api/get_balance?username=Sample123** to verify balance. 5. Send **GET** request **to {baseURL}/api/get_balance?username=user@456!** to verify balance.
Expected Result:	Response code 200. OK Response message. Transfer successful. The sender balance is reduced by the transfer amount. The recipient balance is increased by the transfer amount.

Test Case ID:	ExBank-FT-15
Title:	Send an amount greater than the sender's balance.
Description:	Verify that the system not allowed transactions if the amount greater than sender's balance.
Precondition:	The local Server is running. Both users exist with a known balance.

Test data:	{
	“from”: “Sample123”,
	“to”: “user@456!”,
	“transfer_amount”: “600”
	}

Test steps:	1. Send POST request to **{baseURL}/api/send** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send GET request **to {baseURL}/api/get_balance?Username=Sample123** to verify balance. 5. Send GET request **to {baseURL}/api/get_balance?username=user@456!** to verify balance.
Expected Result:	Response code 400. Bad Request Response message Insufficient funds. The sender balance is unchanged The recipient balance is unchanged.

Test Case ID:	ExBank-FT-16
Title:	Send a zero amount.
Description:	Verify that system not allowed transactions with zero amount.
Precondition:	The local Server is running. Both users exist with a known balance.

Test data:	{
	“from”: “Sample123”,
	“to”: “user@456!”,
	“transfer_amount”: “0”
	}

Test steps:	1. Send POST request to **{baseURL}/api/send** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send GET request **to {baseURL}/api/get_balance?username=Sample123** to verify balance. 5. Send GET request **to {baseURL}/api/get_balance?username=user@456!** to verify balance.
Expected Result:	Response code 400. Bad Request. Response message. Invalid amount The sender balance is unchanged The recipient balance is unchanged.

Test Case ID:	ExBank-FT-17
Title:	Send a negative amount and ensure it fails.
Description:	Verify that the system not allow transactions with negative amounts.
Precondition:	The local Server is running. Both users exist with a known balance.

Test data:	{
	“from”: “Sample123”,
	“to”: “user@456!”,
	“transfer_amount”: “-600”
	}

Test steps:	1. Send POST request to **{baseURL}/api/send** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send GET request **to {baseURL}/api/get_balance?username=Sample123** to verify balance. 5. Send GET request **to {baseURL}/api/get_balance?username=user@456!** to verify balance.
Expected Result:	Response code 400. Bad request. Response message Invalid amount. The sender balance is unchanged The recipient balance is unchanged.

Test Case ID:	ExBank-FT-18
Title:	Send from a non-existing user.
Description:	Verify that the system does not allow transactions from non-existing users.
Precondition:	The local Server is running. One user is created and known balance.

Test data:	{
	“to”: “user@456!”,
	“transfer_amount”: “600”
	}

Test steps:	1. Send POST request to **{baseURL}/api/send** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send GET request **to {baseURL}/api/get_balance?username=user@456!** to verify the balance.
Expected Result:	Response code 404. Not Found Response message Sender not found. The recipient balance is unchanged.

Test Case ID:	ExBank-FT-19
Title:	Send to non-existing user.
Description:	Verify that the system does not allow transactions to non-existing users.
Precondition:	The local server is running. One users is created and known balance.

Test data:	{
	“from”: “Sample123”,
	“transfer_amount”: “600”
	}

Test steps:	1. Send POST request to **{baseURL}/api/send** with data in the body from test data. 2. Verify the response status code. 3. Verify the response message. 4. Send **GET** request **to {baseURL}/api/get_balance?username=Sample123** to verify balance.
Expected Result:	Response code 404. Not Found. Response message Recipient not found. The sender balance is unchanged.

Non-Functional test cases:

Required testing environment:

Node.js for local server.

K6 library.

1. Performance:

Test Case ID:	ExBank-NFT-1
Title:	Measure the response time for “Create_user” under a load of 1100 concurrent requests.
Description:	This test will help assess the system's performance under high-load conditions.
Precondition:	The local server is running. K6 library installed. Ensure no other heavy load operations are running on the system during the test.
Test data:	
Test steps:	1. Run VsCode. 2. Execute the load test with the command “k6 run ExBank_NFT-1.js” 3. Analyse the Result.
Expected Result:	The system should handle 1100 concurrent requests within the specified duration. The response time should be within acceptable limits.

Test Case ID:	ExBank-NFT-2
Title:	Measure the response time for “deposit” and “withdraw” under heavy load.
Description:	This test will help to measure the response time for “deposit “ and “withdraw” endpoints when subjected to heavy load conditions.
Precondition:	The local server is running. K6 library installed. Ensure no other heavy load operations are running on the system during the test.
Test data:	
Test steps:	1. Run VsCode. 2. Execute the load test with the command “k6 run ExBank_NFT-2.js” 3. Analyse the Result.
Expected Result:	The system should handle the load without any significant performance degradation. The response time should be for “deposit” and “ withdraw” within acceptable limits.

Test Case ID:	ExBank-NFT-3
Title:	Measure the response time for “get_balance” for 10,000 users.
Description:	This test will help to measure the response time for “get_balance” endpoints when subjected to heavy load conditions.
Precondition:	The local server is running. K6 library installed. Ensure no other heavy load operations are running on the system during the test.
Test data:	
Test steps:	1. Run VsCode 2. Execute the load test with the command “k6 run ExBank_NFT-3.js” 3. Analyse the Result.
Expected Result:	The system should handle the load without any significant performance degradation. The response time should be for “get_balance” within acceptable limits.

Test Case ID:	ExBank-NFT-4
Title:	Test the performance of “send” under concurrent transactions.
Description:	This test will help to measure the response time for “send” endpoints when subjected to heavy load conditions.
Precondition:	The local server is running. K6 library installed. Ensure no other heavy load operations are running on the system during the test.
Test data:	
Test steps:	1. Run VsCode 2. Execute the load test with the command “k6 run ExBank_NFT-4.js” 3. Analyse the Result.
Expected Result:	The system should handle the load without any significant performance degradation. The response time should be for “send” within acceptable limits.

2. Scalability:

Test Case ID:	ExBank-NFT-5
Title:	Evaluate system behaviour as the number of users increases from 1000 to 100,000.
Description:	
Precondition:	The local server is running. K6 library installed.

	Ensure no other heavy load operations are running on the system during the test.
Test data:	
Test steps:	1. Run VsCode 2. Execute the load test with the command “k6 run ExBank_NFT-5.js” 3. Analyse the Result.
Expected Result:	The system should handle the load without any significant performance degradation.