Bank API Documentation

Author: Vlad Litvak

Table of Contents

Legend:

Function Name
Parameter
Parameter Requirement
JSON key
JSON value
Inner JSON

Info:

- All returned JSONs will have an "Error" key. If its value is false, the JSON is a success response. If its value is true, the JSON is an error response
- ALWAYS check that the "Error" value is false before attempting to extract other expected values from a returned JSON
- If a parameter requirement is not met, the returned JSON will be an error response
- JSON values with asterisks are variable, but will always be of the type specified between the asterisks
    - i.e.
    - "*string*" may be "Transaction Failed: Insufficient Funds"
    - *boolean* may be false
    - "$*number with at least one digit before the decimal and exactly two digits after the decimal*" may be "$175.70"
- The database schema sets a maximum account balance of $999,999,999,999.99. Therefore, initial deposits, deposits, withdraws, and transfer amounts may not exceed this value
- Due to the interweaving use of multiple formatted standards like SQL, JDBC, and JSON: apostrophe's, quotes, backslashes, and semicolons are illegal characters across the service. This means they cannot be included in values for names, passwords, addresses, account names, etc.
- JSONs are case-sensitive
    - Boolean values must be written in lower case (true or false NOT True or False)
    - JSON key strings will be written exactly as described ("Address" is not the same as "address")
- "Time of Transaction" values are in UTC (Coordinated Universal Time)

Register User

Description:
- Attempts to register a new user

Input JSON:
- key:value pairs:
  - "Function":"Register User"
  - "First Name":"*string*"
  - "Last Name":"*string*"
  - "Password":"*string*"
  - "Address":"*string*"

- Example:
  {"Function":"Register User","First Name":"John","Last Name":"Doe","Password":"SecurePassword","Address":"739 Warriors Way, San Francisco, CA 94158"}

Parameters:
- String First Name - The new user's first name
  - max length: 30 characters, cannot contain ' " ; \
- String Last Name - The new user's last name
  - max length: 30 characters, cannot contain ' " ; \
- String Password - The new user's password
  - max length: 30 characters, cannot contain ' " ; \
- String Address - The new user's address
  - max length: 100 characters, cannot contain ' " ; \

Returned JSON:
- Success Response (user created)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "First Name":"*string*"
    - "Last Name":"*string*"

  - Example:
    {"Error":false,"User ID":25371,"First Name":"John","Last Name":"Doe"}

- Error Response (user not created)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
    {"Error":true,"Error Message":"Unable to create user: First name cannot be more than 30 characters or use apostrophes, quotes, backslashes, or semicolons"}

Login

Description:

- Attempts to send a login request
- A successful login request means that the request was processed, to check whether the login was successful, see the “Login Successful” value
- An error response should NEVER be treated as a successful login

Input JSON:

- key:value pairs:
    - "Function":"Login"
    - "User ID":*integer*
    - "Password":"*string*"

- Example:
  {"Function":"Login","User ID":92853,"Password":"password123"}

Parameters:

- int User ID - The customer's user ID
- String Password - The customer's password

Returned JSON:

- Success Response (login request processed)
    - key:value pairs:
        - "Error":false
        - "User ID":*integer*
        - "Login Successful":*boolean*

    - "Login Successful" will be true if the user ID and password match, otherwise it will be false

    - Example:
      {"Error":false,"User ID":92853,"Login Successful":false}

- Error Response (login request not processed)
    - key:value pairs:
        - "Error":true
        - "Error Message":"*string*"

    - Example:
      {"Error":true,"Error Message":"Unable to determine login for User #92853"}

Change Name

Description:
- Attempts to change a user's name

Input JSON:
- key:value pairs:
  - "Function":"Change Name"
  - "User ID":*integer*
  - "Password":"*string*"
  - "New First Name":"*string*"
  - "New Last Name":"*string*"

- Example:
  ```
  {"Function":"Change Name","User ID":66493,"Password":"password123","New First Name":"John","New Last Name":"Smith"}
  ```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- String New First Name - The customer’s new first name
  - max length: 30 characters, cannot contain ' " ; \
- String New Last Name - The customer’s new last name
  - max length: 30 characters, cannot contain ' " ; \

Returned JSON:
- Success Response (name changed)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "New First Name":"*string*"
    - "New Last Name":"*string*"

  - Example:
    ```
    {"Error":false,"User ID":66493,"New First Name":"John","New Last Name":"Smith"}
    ```

- Error Response (name not changed)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
    ```
    {"Error":true,"Error Message":"Unable to change name for User #66493: First name cannot be more than 30 characters or use apostrophes, quotes, backslashes, or semicolons"}
    ```

Change Address

Description:
- Attempts to change a user's address

Input JSON:
- key:value pairs:
  - "Function":"Change Address"
  - "User ID":*integer*
  - "Password":"*string*"
  - "New Address":"*string*"

- Example:
  ```
  {"Function":"Change Address","User ID":58354,"Password":"WarriorsFan987","New Address":"404 Main Street, San Francisco, CA 94121"}
  ```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- String New Address - The customer’s new address
  - max length: 100 characters, cannot contain ' " ; \

Returned JSON:
- Success Response (address changed)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "New Address":"*string*"

  - Example:
    ```
    {"Error":false,"User ID":58354,"New Address":"404 Main Street, San Francisco, CA 94121"}
    ```

- Error Response (address not changed)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
    ```
    {"Error":true,"Error Message":"Unable to change address for User #58354: User ID and password do not match"}
    ```

Change Password

Description:
- Attempts to change a user's password

Input JSON:
- key:value pairs:
    - "Function":"Change Address"
    - "User ID":*integer*
    - "Old Password":"*string*"
    - "New Password":"*string*"

- Example:
  {"Function":"Change Password","User ID":43112,"Old Password":"oldPASSWORD99","New Password":"NewSecurePassword44"}

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Old Password - The customer's old password
  - must match user ID
- String New Password - The customer's new password
  - max length: 30 characters, cannot contain ' " ; \

Returned JSON:
- Success Response (password changed)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "Password Changed":true

  - "Password Changed" will always be true in the success response. It is a boolean, not a string containing the new password, because a user's password will never be sent out by any API call

  - Example:
    {"Error":false,"User ID":43112,"Password Changed":true}

- Error Response (password not changed)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
    {"Error":true,"Error Message":"Unable to change password for User #43112: New password must be different than the current password"}

Open Account

Description:
- Attempts to create a new bank account for a user

Input JSON:
- key:value pairs:
  - "Function":"Open Account"
  - "User ID":*integer*
  - "Password":"*string*"
  - "Account Name":"*string*"
  - "Account Type":"*string*"
  - "Initial Deposit":*number with at least one digit before the decimal and exactly two digits after the decimal*
- Example:
  {"Function":"Open Account","User ID":773659,"Password":"bankpassword","Account Name":"Chris Checking Account","Account Type":"C","Initial Deposit":14055.90}

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- String Account Name - The new account's name
  - max length: 50 characters, cannot contain ' " ; \
- String Account Type – The account type
  - must be either "C" for Checking or "S" for Savings
- double Initial Deposit - The initial deposit for the new account
  - must be formatted as a number with two digits after the decimal (i.e. 100.00)
  - must be greater than or equal to zero and less than one trillion

Returned JSON:
- Success Response (account created)
  - key:value pairs:
    - "Error":false
    - "Account ID":*integer*
    - "User ID":*integer*
    - "Account Name":"*string*"
    - "Account Type":"*string*"
    - "Balance":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"
  - "Account Type" will always be either "Checking" or "Savings"
  - Example:
    {"Error":false,"Account ID":2295722,"User ID":773659,"Account Name":"Chris Checking Account","Account Type":"Checking","Balance":"$14055.90"}
- Error Response (account not created)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"
  - Example:
    {"Error":true,"Error Message":"Unable to create account (Chris Checking Account) for User #773659: Initial deposit cannot be negative"}

Change Account Name

Description:
- Attempts to change an account’s name

Input JSON:
- key:value pairs:
    - "Function":"Change Account Name"
    - "User ID":*integer*
    - "Password":"*string*"
    - "Account ID":*integer*
    - "New Account Name":"*string*"

- Example:
```
{"Function":"Change Account Name","User ID":75927,"Password":"RedGreenBlue","Account ID":673054,"New Account Name":"College Savings for Julie"}
```

Parameters:
- int User ID - The customer's user ID
    - must be valid
- String Password - The customer's password
    - must match user ID
- int Account ID - The account ID
    - must be active and owned by the specified user
- String New Account Name - The account’s new name
    - max length: 50 characters, cannot contain ' " ; \

Returned JSON:
- Success Response (address changed)
    - key:value pairs:
        - "Error":false
        - "User ID":*integer*
        - "Account ID":*integer*
        - "New Account Name":"*string*"

    - Example:
```
{"Error":false,"User ID":75927,"Account ID":673054,"New Account Name":"College Savings for Julie"}
```

- Error Response (address not changed)
    - key:value pairs:
        - "Error":true
        - "Error Message":"*string*"

    - Example:
```
{"Error":true,"Error Message":"Unable to change name for Account #673054: User ID and password do not match"}
```

Close Account

Description:
- Attempts to close a user's account, withdraws remaining account balance

Input JSON:
- key:value pairs:
    - "Function":"Close Account"
    - "User ID":*integer*
    - "Password":"*string*"
    - "Account ID":*integer*

- Example:
  {"Function":"Close Account","User ID":46033,"Password":"asdfg09876","Account ID":286736}

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- int Account ID - The account ID
  - must be active and owned by the specified user

Returned JSON:
- Success Response (account closed)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "Account ID":*integer*
    - "Closed":true

  - "Closed" will always be true in the success response

  - Example:
    {"Error":false,"User ID":46033,"Account ID":286736,"Closed":true}

- Error Response (account not closed)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
    {"Error":true,"Error Message":"Unable to close Account #286736 (User #46033): User has no active account with this account number"}

Withdraw

Description:
- Attempts to send a withdraw request
- A successful withdraw request means that the request was processed, to check whether the transaction went through or not, see the “Status” value

Input JSON:
- key:value pairs:
  - "Function":"Withdraw"
  - "User ID":*integer*
  - "Password":"*string*"
  - "Account ID":*integer*
  - "Amount":*number with at least one digit before the decimal and exactly two digits after the decimal*

- Example:
```
{"Function":"Withdraw","User ID":13743,"Password":"PaSsWoRd","Account ID":605112,"Amount":100.00}
```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- int Account ID - The account to be withdrawn from
  - must be active and owned by the specified user
- double Amount - The amount to be withdrawn from the account
  - must be formatted as a number with two digits after the decimal (i.e. 100.00)
  - must be greater than zero and less than one trillion

Returned JSON:
- Success Response (withdraw request processed)
  - key:value pairs:
    - "Error":false
    - "Transaction ID":*integer*
    - "Transaction Type":"Withdraw"
    - "Amount":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"
    - "Account ID":*integer*
    - "User ID":*integer*
    - "Status":"*string*"
    - "Time of Transaction":"*string formatted as YYYY-MM-DD HH:MM:SS in UTC*"

  - Example:
```
{"Error":false,"Transaction ID":162508014,"Transaction Type":"Withdraw","Amount":"$100.00","Account ID":605112,"User ID":13743,"Status":"Transaction Failed: Insufficient Funds","Time of Transaction":"2020-04-08 08:55:11"}
```

- Error Response (withdraw request not processed)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
```
{"Error":true,"Error Message":"Unable to withdraw $1000000000000.00 from Account #605112: Withdraw amount cannot exceed $999,999,999,999.99"}
```

Deposit

Description:
- Attempts to send a deposit request
- A successful deposit request means that the request was processed, to check whether the transaction went through or not, see the “Status” value

Input JSON:
- key:value pairs:
    - "Function":"Depoist"
    - "User ID":*integer*
    - "Password":"*string*"
    - "Account ID":*integer*
    - "Amount":*number with at least one digit before the decimal and exactly two digits after the decimal*

- Example:
```
{"Function":"Deposit","User ID":33603,"Password":"passwordFORbank","Account
ID":104609,"Amount":0.99}
```

Parameters:
- int User ID - The customer's user ID
    - must be valid
- String Password - The customer's password
    - must match user ID
- int Account ID - The account to be deposited into
    - must be active and owned by the specified user
- double Amount - The amount to be deposited into the account
    - must be formatted as a number with two digits after the decimal (i.e. 100.00)
    - must be more than zero and less than one trillion

Returned JSON:
- Success Response (deposit request processed)
    - key:value pairs:
        - "Error":false
        - "Transaction ID":*integer*
        - "Transaction Type":"Deposit"
        - "Amount":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"
        - "Account ID":*integer*
        - "User ID":*integer*
        - "Status":"*string*"
        - "Time of Transaction":"*string formatted as YYYY-MM-DD HH:MM:SS in UTC*"

    - Example:
```
{"Error":false,"Transaction ID":911657521,"Transaction
Type":"Deposit","Amount":"$0.99","Account ID":104609,"User
ID":33603,"Status":"Transaction Complete","Time of Transaction":"2020-02-13
17:20:55"}
```

- Error Response (deposit request not processed)
    - key:value pairs:
        - "Error":true
        - "Error Message":"*string*"

    - Example:
```
{"Error":true,"Error Message":"Unable to depost $0.99 into Account #104609:
User ID and password do not match"}
```

Transfer

Description:
- Attempts to send a transfer request
- A successful transfer request means that the request was processed, to check whether the transaction went through or not, see the “Status” value

Input JSON:
- key:value pairs:
    - "Function":"Transfer"
    - "User ID":*integer*
    - "Password":"*string*"
    - "Source Account ID":*integer*
    - "Destination Account ID":*integer*
    - "Amount":*number with at least one digit before the decimal and exactly two digits after the decimal*

- Example:
```
{"Function":"Transfer","User ID":33652,"Password":"abcdefg1234","Source Account ID":4104586,"Destination Account ID":152431,"Amount":15999.99}
```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- int Source Account ID - The account to be withdrawn from
  - must be active and owned by the specified user
- int Destination Account ID - The account to be deposited into
  - must be active
  - must be different than the source account
- double Amount - The amount to be transferred from the source account to the destination account
  - must be formatted as a number with two digits after the decimal (i.e. 100.00)
  - must be more than zero and less than one trillion

Returned JSON (Cont. on next page)

Transfer (Cont.)

Returned JSON:

- Success Response (transfer request processed)
  - key:value pairs:
    - "Error":false
    - "Transaction ID":*integer*
    - "Transaction Type":"Transfer"
    - "Amount":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"
    - "Source Account ID":*integer*
    - "Destination Account ID":*integer*
    - "User ID":*integer*
    - "Status":"*string*"
    - "Time of Transaction":"*string formatted as YYYY-MM-DD HH:MM:SS in UTC*"
  - Example:

    ```
    {"Error":false,"Transaction ID":329830315,"Transaction
    Type":"Transfer","Amount":"$15999.99","Source Account ID":4104586,"Destination
    Account ID":152431,"User ID":33652,"Status":"Transaction Complete","Time of
    Transaction":"2019-11-29 14:23:09"}
    ```

- Error Response (transfer request not processed)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"
  - Example:

    ```
    {"Error":true,"Error Message":"Unable to transfer $25.50 from Account #400732
    to Account #165012: User #58487 has no active accounts with the Account Number
    400732"}
    ```

User Info

Description:
- Attempts to get a user's personal information

Input JSON:
- key:value pairs:
  - "Function":"User Info"
  - "User ID":*integer*
  - "Password":"*string*"

- Example:
  ```
  {"Function":"User Info","User ID":80341,"Password":"MyPassword115"}
  ```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID

Returned JSON:
- Success Response (user info returned)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "First Name":"*string*"
    - "Last Name":"*string*"
    - "Address":"*string*"

  - Example:
    ```
    {"Error":false,"User ID":80341,"First Name":"Stephen","Last
    Name":"Johnson","Address":"747 Avian Drive, Los Angeles, CA 90210"}
    ```

- Error Response (user info not returned)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
    ```
    {"Error":true,"Error Message":"Cannot get user info for User #80341: User ID
    and password do not match"}
    ```

User Account Summary

Description:
- Attempts to get a list of a user's active accounts

Input JSON:
- key:value pairs:
  - "Function":"User Account Summary"
  - "User ID":*integer*
  - "Password":"*string*"
  - "Include Inactive":*boolean*

- Example:
```
{"Function":"User Account Summary","User
ID":17374,"Password":"password1","Include Inactive":true}
```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- boolean Include Inactive – Whether to include inactive accounts in the list
  - if true, inactive accounts will be included, otherwise they will not

Returned JSON:
- Success Response (user’s active accounts returned)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "Accounts":[*JSON array of Account JSONs*]

  - For info on Account JSONs, see page 21

  - If the user has no active accounts, "Accounts" will be an empty JSON array (e.g. [])

  - Example:
```
{"Error":false,"User ID":17374,"Accounts":[{"Account ID":946832,"Account
Name":"College Funds","Account Type":"Savings","Account
Status":"Active","Balance":"$1500.00"},{"Account ID":946421,"Account
Name":"David - Checking","Account Type":"Checking","Account
Status":"Inactive","Balance":"$0.00"}]}
```

- Error Response (user’s active accounts not returned)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
```
{"Error":true,"Error Message":"Unable to get summary of User #17374's accounts:
User ID and password do not match"}
```

User Transaction History

Description:
- Attempts to get a user's transaction history in order of most recent transaction first

Input JSON:
- key:value pairs:
  - "Function":"User Transaction History"
  - "User ID":*integer*
  - "Password":"*string*"
  - "Limit":*integer*

- Example:
```
{"Function":"User Transaction History","User
ID":86364,"Password":"BankPassword","Limit":0}
```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- int Limit - The maximum number of transactions to be shown
  - if limit is 0 or less, no limit will be applied

Returned JSON (Cont. on next page)

User Transaction History (Cont.)

Returned JSON:

- Success Response (user transaction history returned)
  - key:value pairs:
    - "Error":false
    - "User ID":*integer*
    - "Transactions":[*JSON array of Transaction JSONs*]

  - For info on Transaction JSONs, see page 22

  - "Transactions" will never contain a Transaction JSON with a "Transaction Type" of "Incoming Transfer" because users can only make transfers from an account they own to another account

  - If the user has made no transactions, "Transactions" will be an empty JSON array (e.g. [])

  - Example:
{"Error":false,"User ID":86364,"Transactions":[{"Transaction ID":105783564,"Transaction Type":"Withdraw","Amount":"$1000.00","Account ID":438608,"User ID":86364,"Status":"Transaction Failed: Insufficient Funds","Time of Transaction":"2020-04-09 09:46:30"},{"Transaction ID":105783391,"Transaction Type":"Outgoing Transfer","Amount":"$780.00","Account ID":438608,"Destination Account ID":639644,"User ID":86364,"Status":"Transaction Complete","Time of Transaction":"2020-04-06 13:02:31"},{"Transaction ID":105783022,"Transaction Type":"Deposit","Amount":"$25.01","Account ID":438608,"User ID":86364,"Status":"Transaction Complete","Time of Transaction":"2020-04-01 20:28:59"}]}

- Error Response (user transaction history not returned)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
{"Error":true,"Error Message":"Cannot get transaction history for User #86364: User ID and password do not match"}

Account Transaction History

Description:
- Attempts to get an account’s transaction history in order of most recent transaction first

Input JSON:
- key:value pairs:
  - "Function":"Account Transaction History"
  - "User ID":*integer*
  - "Password":"*string*"
  - "Account ID":*integer*
  - "Limit":*integer*

- Example:
```
{"Function":"Account Transaction History","User ID":64366,"Password":"awsdjikl","Account ID":129739,"Limit":10}
```

Parameters:
- int User ID - The customer's user ID
  - must be valid
- String Password - The customer's password
  - must match user ID
- int Account ID - The account ID
  - must be owned by the specified user, can be inactive
- int Limit - The maximum number of transactions to be shown
  - if limit is 0 or less, no limit will be applied

Returned JSON:
- Success Response (account transaction history returned)
  - key:value pairs:
    - "Error":false
    - "Account ID":*integer*
    - "Transactions":[*JSON array of Transaction JSONs*]

  - For info on Transaction JSONs, see page 22

  - If the account has no transactions associated with it, "Transactions" will be an empty JSON array (e.g. [])

  - Example:
```
{"Error":false,"Account ID":129739,"Transactions":[{"Transaction ID":973748636,"Transaction Type":"Incoming Transfer","Amount":"$100.00","Account ID":129739,"Source Account ID":408535,"User ID":36894,"Status":"Transaction Complete","Time of Transaction":"2020-04-02 08:51:31"},{"Transaction ID":973748212,"Transaction Type":"Deposit","Amount":"$55.66","Account ID":129739,"User ID":64366,"Status":"Transaction Complete","Time of Transaction":"2020-03-15 15:47:09"},{"Transaction ID":973747527,"Transaction Type":"Outgoing Transfer","Amount":"$19.48","Account ID":129739,"Destination Account ID":846822,"User ID":64366,"Status":"Transaction Complete","Time of Transaction":"2020-03-13 12:00:17"}]}
```

- Error Response (account transaction history not returned)
  - key:value pairs:
    - "Error":true
    - "Error Message":"*string*"

  - Example:
```
{"Error":true,"Error Message":"Cannot get transaction history for Account #129739: User ID and password do not match"}
```

Account JSON

- key:value pairs:
  - "Account ID":*integer*
  - "Account Name":"*string*"
  - "Account Type":"*string*"
  - "Account Status":"*string*"
  - "Balance":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"

- "Account Type" will always be either "Checking" or "Savings"
- "Account Status" will always be either "Active" or "Inactive"

- Example:
  {"Account ID":946832,"Account Name":"College Funds","Account Type":"Savings","Account Status":"Active","Balance":"$1500.00"}

Transaction JSON

- Deposit or Withdraw
    - key:value pairs:
        - "Transaction ID":*integer*
        - "Transaction Type":"*String*"
        - "Amount":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"
        - "Account ID":*integer*
        - "User ID":*integer*
        - "Status":"*string*"
        - "Time of Transaction":"*string formatted as YYYY-MM-DD HH:MM:SS in UTC*"
    - "Transaction Type" will be either "Deposit" or "Withdraw", accordingly
    - Example:
      {"Transaction ID":105783564,"Transaction Type":"Withdraw","Amount":"$1000.00","Account ID":438608,"User ID":86364,"Status":"Transaction Failed: Insufficient Funds","Time of Transaction":"2020-04-09 09:46:30"}
- Outgoing Transfer
    - key:value pairs:
        - "Transaction ID":*integer*
        - "Transaction Type":"Outgoing Transfer"
        - "Amount":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"
        - "Account ID":*integer*
        - "Destination Account ID":*integer*
        - "User ID":*integer*
        - "Status":"*string*"
        - "Time of Transaction":"*string formatted as YYYY-MM-DD HH:MM:SS in UTC*"
    - Example:
      {"Transaction ID":105783391,"Transaction Type":"Outgoing Transfer","Amount":"$780.00","Account ID":438608,"Destination Account ID":639644,"User ID":86364,"Status":"Transaction Complete","Time of Transaction":"2020-04-06 13:02:31"}
- Incoming Transfer
    - key:value pairs:
        - "Transaction ID":*integer*
        - "Transaction Type":"Incoming Transfer"
        - "Amount":"$*number with at least one digit before the decimal and exactly two digits after the decimal*"
        - "Account ID":*integer*
        - "Source Account ID":*integer*
        - "User ID":*integer*
        - "Status":"*string*"
        - "Time of Transaction":"*string formatted as YYYY-MM-DD HH:MM:SS in UTC*"
    - Example:
      {"Transaction ID":105784109,"Transaction Type":"Incoming Transfer","Amount":"$38.95","Account ID":438608,"Source Account ID":163739,"User ID":74199,"Status":"Transaction Complete","Time of Transaction":"2020-04-15 20:29:28"}