NENO SACCO C2B Transactions Capturing System Documentation

This is the documentation for the **C2B transaction capturing system** developed using Python (Django). The system works by simply listening for transactions made to the *Neno Sacco Paybill Number* and storing it in a database for future reference. To achieve this the system has to get a callback from the Mpesa API after successful payment.

Mpesa C2B Model

This system has been implemented following the Mpesa C2B model. The flow of a C2B transaction is as follows:

Mpesa C2B Transaction Flow

Below is an explanation of the Mpesa C2B process:

- A customer sends a payment request to your paybill from their phone
- MPesa receives the request and validates it internally first
- MPesa checks if you have enabled **External Validation** for the paybill receiving the request
- If External Validation is enabled:
    - MPesa first sends a **Validation request** to the Validation URL registered in the system (3rd party) with the payment details.
    - The 3rd Party validates the request and sends an appropriate response to MPesa. This response must be received within a given time period or MPesa marks the endpoint system as unreachable. The response is either to complete or cancel the payment:
    - MPesa receives the response and processes the transaction accordingly:
    - If you had chosen to complete the transaction, MPesa sends a **Confirmation request** to your Confirmation URL with the details of the completed transaction. The transaction is then complete. Hence you receive **two** API calls on your system.
    - If you had chosen to cancel the payment, MPesa simply cancels the transaction and no other request is sent. The transaction is then completed and closed.
- If External Validation is **disabled**, MPesa automatically completes the transaction, and if the transaction is a success, MPesa sends a **Confirmation** request to the Confirmation URL registered in the system. This is the **only** API call received on your end.
- If External Validation is **enabled**, but for some reason MPesa could **not** reach your endpoint to validate the transaction within the stipulated time period (usually < 8 seconds), or no response was received by the time MPesa terminates the request, it checks on the default action value saved during registration of the URLs. If the default action was set to **Completed**, MPesa automatically completes the transaction and also tries to send a Confirmation request to your

other endpoint. If the default action was set to **Cancelled**, MPesa simply cancels the transaction. The transaction is then complete.

- If no URLs are registered in the system, MPesa automatically completes the request. MPesa then sends an SMS notification to both the customer and paybill owner with the results of the transaction as usual.
- If the external notifications fails to sent, you can check on the MPesa Org portal and cross-check against received callbacks. The portal has all the payments made available, whether you received the callback or not.

Setting Up the Project

Install Virtual Enviroment

If you are on linux you can simply achieve this by running

```
python3 -m venv venv
```

Install required modules

First update pip to latest version

```
pip install --upgrade pip
```

Then install the requirements

```
pip3 install -r requirements.txt
```

Set up the required secret variables and API keys

Open the .env file and add all required variables. This include:

```
BASE_URL = ''
CONSUMER_KEY = ''
CONSUMER_SECRET = ''
SHORTCODE = ''

AT_USERNAME = ''
AT_API_KEY = ''
```

After that make and run migrations

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Start server

```
python manage.py runserver
```

Project Implementation

Project File Strcture

├── base │ ├── `__init__.py` │ ├── `__pycache__` │ ├── settings.py │ ├── urls.py │ └── wsgi.py ├── c2b │ ├── admin.py │ ├── apps.py │ ├── `__init__.py` │ ├── logic.py │ ├── migrations │ ├── models.py │ ├── `__pycache__` │ ├── templates │ ├── tests.py │ ├── urls.py │ └── views.py ├── db.sqlite3 ├── manage.py └── requirements.txt

> This Django project has one app **c2b** which is responsible for capturing the transactions, notify a customer who initiated the transaction on a successful transaction and store transaction in a database. logic.py - Is the main module in the app. It contains a class which implements the logic for capturing the transactions and sending notifications on successful transactions. The class contains several functions like **access_token**, **register_url** and **notify**

Project Explanation

According to the daraja api documentation so as to successfully implement the C2B model as in the case of this project you must follow the process below

1. Authentication
2. Register the callback URLs

1. Authentication

In order to make any API call to Mpesa, the app need to be authenticated first. Safaricom provides an OAuth API which generates the tokens for authenticating your API calls. This is the first API the app will engage with in the set of APIs available because all the other APIs require authentication information from this API so as to work. To better understand this; imagine this step as a reception desk probably in a hotel where you are issued a key(access token) which you can use to access a room. For you to be issued the keys you must prove that you have the permission to access that room maybe by providing information about yourself which can be checked to determine if you the permission to access the room. As explained in this analogy, you are the app, the reception is the Safaricom authentication api that issues you with access tokens(keys) so that you can interact with other APIs(access rooms). The information you provide at the reception desk to prove if you have permission to access a room are your app's **Consumer key** and **Consumer Secret** which can be found under the **Keys** tab from **My apps** here

So as to authenticate the app hits the https://api.safaricom.co.ke/oauth/v1/generate?grant_type=client_credentials with a base64 encoded string of the app's **client key** and **client secret** (*Base-64 encoding of **Consumer Key + ":" + Consumer Secret***) as shown below. In this project, this is simply implemented using requests' *HTTPBasicAuth* module. Where we only pass in the endpoint(URL) where the authentication will happen, the client key and secret key.

```
from django.conf import settings
import requests
from requests.auth import HTTPBasicAuth

class  SetCallback:
    # Get credentials from settings
    consumer_key = settings.CONSUMER_KEY
    consumer_secret = settings.CONSUMER_SECRET
    shortcode = settings.SHORTCODE
    base_url = settings.BASE_URL # Callback base url

    """
    get_token hits the authentication endpoint with Base-64 encoding of Consumer Key +
    returns an access token
    """

    def  get_token(self):
        authentication_URL = "https://api.safaricom.co.ke/oauth/v1/generate?grant_type
        response = requests.get(authentication_URL, auth=HTTPBasicAuth(self.consumer_k
        if response.status_code == 200:
            return response.json()['access_token']
```

On a successful authentication, a JSON response is returned. The response contains the access token and the timeout period in which the access token will be valid. The access token is retrieved by:

```
response.json()['access_token']
```

> The access_token expires in 3600 seconds or 1 hour

Once the access token has been received successfully, the next step (**Registering Callback Urls**) is initiated.

2. Registering Callback URLs

In this step, you inform the Mpesa API on where to send payment notifications for payments to the paybill. The URLs are used by the C2B payment simulation API for sending the transaction details to the app for processing and storing to a database. There are two URLs required for RegisterURL API: **Validation URL** and **Confirmation URL**. The reason for two of them is due to the nature of C2B API calls as explained above on the Mpesa C2B model explanation. In this project, we are only interested in only one the (**Confirmation URL**) Registering URLs is achieved by a post request to the url registration callback endpoint (https://api.safaricom.co.ke/mpesa/c2b/v1/registerurl). The post request contains header which is a dictionary of extra information sent with the request and a body which is the data sent to the API. Registering callback urls for this app is implemented as below.

```
from django.conf import settings
import requests
from requests.auth import HTTPBasicAuth

class  SetCallback:
    # Get credentials from settings
    consumer_key = settings.CONSUMER_KEY
    consumer_secret = settings.CONSUMER_SECRET
    shortcode = settings.SHORTCODE
    base_url = settings.BASE_URL # Callback base url

    """
    get_token hits the authentication endpoint with Base-64 encoding of Consumer Key +
    returns an access token
    """

    def  get_token(self):
        authentication_URL = "https://api.safaricom.co.ke/oauth/v1/generate?grant_type
        response = requests.get(authentication_URL, auth=HTTPBasicAuth(self.consumer_k
        if response.status_code == 200:
            return response.json()['access_token']

    """"
    register_url registers the callback urls (Confirmation and Validation URLs) and se
    in case the validation URL is unreachable.
    """
    def  register_url(self, access_token):
        url_register = 'https://api.safaricom.co.ke/mpesa/c2b/v1/registerurl'
        headers = {'Authorization': 'Bearer %s' % access_token, 'content-type': 'appli
        body = {
            "ShortCode": "%s" % self.shortcode,
            "ResponseType": "Completed",
            "ConfirmationURL": "%s/api/v1/c2b/confirmation" % self.base_url,
        }

        response = requests.post(url_register, headers=headers, json=body)
        if response.status_code == 200:
            return response.json()
```

The access token returned by the **get_token** function is passed to **register_url**. Headers included in the post request consist of a ***bearer token*** for authorization and a ***content-type*** used to indicate the media **type** of the resource. The body consists of the organization paybill number, A ResponseType which specifies what the Mpesa API should incase it fails to reach the application's **Validation Callback**. If the Mpesa API fails to reach the validation callback it can either **complete** or **cancel** the transaction. These options are what is specified in the body as ResponseType. The Callback and Validation URLs are also sent in the body to specify where Mpesa API should send the validation and confirmation details. The **register_url** function returns a response as the one below

```
{
  "ConversationID": "",
  "OriginatorCoversationID": "",
  "ResponseDescription": "success"
}
```

If any other data than the above is in the response then the register url will have failed. So as to be certain if registering URLs failed or was successful, check the response's *response_status*. A **200** means it was successful.

What to Note about the callback urls

> **Note:** Use publicly available (Internet-accessible) IP addresses or domain names

> **Note:** Do not use the words **MPesa, M-Pesa, Safaricom** or any of their variants in either upper or lower cases in your URLs, the system filters these URLs out and blocks them. Of course, any Localhost URL will be refused.

> **Note:** Do not use public URL testers e.g. mockbin or requestbin **especially** on production, they are also blocked by the API.

3. Notify Clients on a Successful Payment to the Paybill

After a successful transaction by the client. The client is notified of the complete notification through an SMS. Notification through SMS is achieved by integrating the **Africas Talking** SMS API into the app. Sending SMS using the Africas Talking API is as easy as defining the required details for SMS sending. This include the Africas Talking **app username** and **API key**, the **recepient's phone number**, **message** to be sent.

Code explanation

So as to send the SMSs using the Africas Talking API, first we have to install the SDK from pip using ***pip install africastalking***. The SDK is then imported into the project by ***import africastalking***. This brings in Africa's Talking's SDK into the app and now we need to get authenticated with the service. We will define a user name and API key that will serve as our credentials. We need to get to use the Africa's Talking service and to do that we need to start up with it. We then asiign the `SMS` class to the `sms` variable. We can now use all the functions that come along with the `SMS` class! Next up, we need to define some variables that will hold the data we will be sending through to the API. The variables include:

- `recipients` - The phone number where the message will be sent (should be an array of numbers as strings)
- `message` - The message to be sent
- `sender` - This holds the senderId or shortcode of the sender

The last part is the actual sending of the SMS. Here, the SMS sending code is wrapped in a `try-catch` statement that makes sure that if the code runs in the `try` part, the response from the Africa's Talking servers and in the event that it fails, the `catch` statement will provide us with an error message that tells us exactly where things went wrong.

In the project we extract the amount paid, the customers name, the account paid to and the customers phone number from the response of the confirmation callback as shown below

```
mpesa_body =request.body.decode('utf-8')
mpesa_payment = json.loads(mpesa_body)

# Get payment details from response
first_name = mpesa_payment['FirstName']
amount = mpesa_payment['TransAmount']
account_no = mpesa_payment['BillRefNumber']
phone_number = mpesa_payment['MSISDN']
```

The actual sending of SMS is implemented by the class below.

```
class  Notify:
    # Get the Africas Talking credentials
    AT_username = settings.AT_USERNAME
    AT_api_key = settings.AT_API_KEY

    def  __init__(self):
        # Initialising the Africas Talking SDK
        africastalking.initialize(self.AT_username, self.AT_api_key)

        # Get the SMS service
        self.sms = africastalking.SMS

    def  notify_customer(self, first_name, amount, account_no, phone_number):
        recipients = [phone_no]
        message = f"Hello {first_name}, we have received your payment of {amount} for

        try:
            response = self.sms.send(message, recipients)
        except  Exception  as e:
            print ('Encountered an error while sending: %s' % str(e))
```

Confirmation Callback

The app has a confirmation endpoint that keeps on listening for any response from the Mpesa API in the case of a successful transaction. Once a transaction has been successful the Mpesa API send details of the transaction to the app's confirmation callback endpoint. The details contains ***Transaction Type***, ***Transaction ID***, ***Transaction Time***, ***Transaction Amount***, ***Paybill Number*** (BusinessShortCode), ***Account Number*** (BillRefNumber), ***Invoice Number***, ***Organisation Account Balance***, ***3rd Party Transaction ID***, ***Client's Phone Number*** (MSISDN), ***Client's First Name***, ***Client's Middle Name***, ***Client's Last Name***. A sample response is as shown below

```
{
  "TransactionType": "",
   "TransID": "LHG31AA5TX",
  "TransTime": "20170816190243",
  "TransAmount": "200.00",
  "BusinessShortCode": "600610",
  "BillRefNumber": "account",
  "InvoiceNumber": "",
  "OrgAccountBalance": "",
  "ThirdPartyTransID": "",
  "MSISDN": "254708374149",
  "FirstName": "John",
  "MiddleName": "",
  "LastName": "Doe"
}
```

TransID

This is MPesa's unique transaction identifier for your transaction. This can be used to search for the transaction later on using the **Transaction Query API**.

TransTime

Simply the time the transaction was completed on MPesa in the format YYYYMMddHHmmss.

TransAmount

The amount transacted by the customer when paying to your paybill/till.

BusinessShortCode

The shortcode to which the customer paid to. This can be used to differentiate payments to different paybills via the same notification URLs.

BillRefNumber

The account number the customer entered on their phone when making the payment. Applicable to PayBill requests.

MSISDN

The phone number from which the payment was made.

FirstName, MiddleName, LastName

The names of the customer under whom the MSISDN above is registered. The First Name and Last Name are usually mandatory. The Middle Name is optional.

In the Confirmation Callback the required data is extracted and stored in the database and a response returned to the API call with either an **accept** or **reject** response. To accept, the send the below JSON is sent making sure the value of **ResultCode** is 0 (zero), but the value of ResultDesc can be any alphanumeric value. This is implemented in the app as shown below

```
from django.shortcuts import render
from django.views.decorators.csrf import csrf_exempt
from c2b.models import C2bPayment
from django.http import JsonResponse

@csrf_exempt
def  confirmation(request):
    mpesa_body =request.body.decode('utf-8')
    mpesa_payment = json.loads(mpesa_body)

    payment = C2bPayment(
        first_name=mpesa_payment['FirstName'],
        last_name=mpesa_payment['LastName'],
        middle_name=mpesa_payment['MiddleName'],
        description=mpesa_payment['TransID'],
        phone_number=mpesa_payment['MSISDN'],
        amount=mpesa_payment['TransAmount'],
        reference=mpesa_payment['BillRefNumber'],
        organization_balance=mpesa_payment['OrgAccountBalance'],
        type=mpesa_payment['TransactionType'],
    )

    payment.save()
    context = {
        "ResultCode": 0,
        "ResultDesc": "Accepted"
    }
    return JsonResponse(dict(context))
```

Displaying Transactions

This is a view that displays all the transactions for the Organisation. The view just pulls data from the database and displays it in a template

```
from django.views.generic import ListView
from c2b.models import C2bPayment

class  TransactionsListView(ListView):
    model = C2bPayment
    template_name = 'transactions.html'
```