Mpesa STKPUSH for WooCommerce Android/ IOS

Using firebase functions

Mpesa STKPUSH for WooCommerce is a payment gateway integration that allows customers to make payments using Mpesa mobile money on a WooCommerce website. This integration works on both Android and IOS platforms.

STKPUSH stands for "Sim Toolkit Push". It is a feature provided by Safaricom, the company behind Mpesa, which allows payment requests to be sent directly to a customer's phone through the SIM card. This means that customers do not have to enter their phone numbers or any other details during the payment process.

With the Mpesa STKPUSH for WooCommerce integration, customers can simply select Mpesa as their payment method during checkout, and a payment request will be sent to their phone via STKPUSH. They will then receive a prompt on their phone to confirm the payment, and once confirmed, the payment will be processed and the order will be marked as paid on the WooCommerce website.

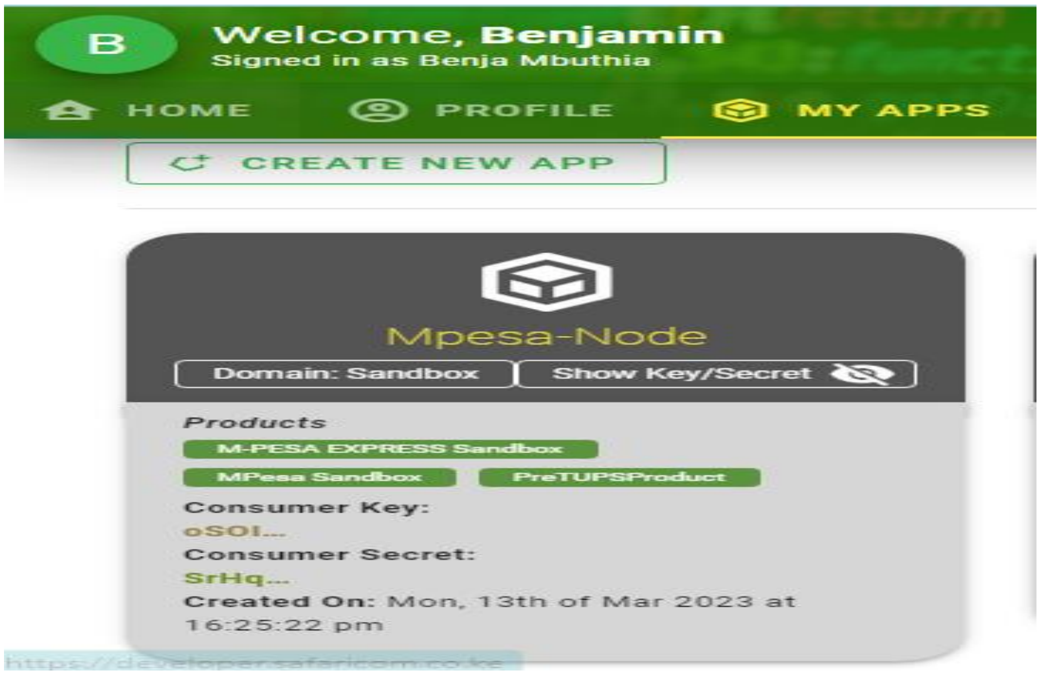
The Mpesa STKPUSH for WooCommerce integration provides a seamless and secure payment experience for customers, and helps to reduce payment delays and errors. It also provides merchants with a reliable and efficient payment solution, helping to increase sales and customer satisfaction.

Prerequisites

1. Have node and npm installed in your machine (Get them here https://nodejs.org/en)
2. Stable internet connection

Create Mpesa app (sandbox)

1. Head over to https://developer.safaricom.co.ke/
2. Create account and create sandbox app there
3. Copy **consumer key** and **consumer secret** and replace them respectively in the mpesa-config.json file SANDBOX_CONSUMER_KEY and SANDBOX_CONSUMER_SECRET



Welcome, Benjamin
Signed in as Benja Mbuthia

Products

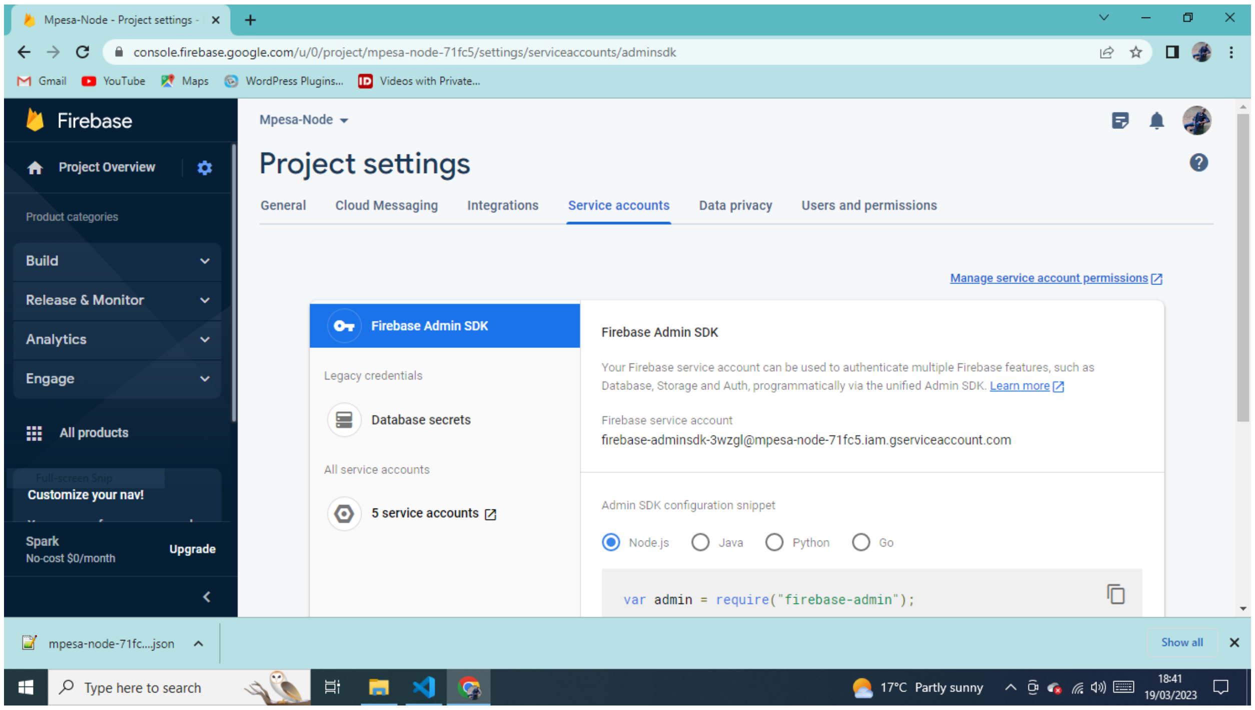
Consumer Key:
oSOI...
Consumer Secret:
SrHq...
Created On: Mon, 13th of Mar 2023 at 16:25:22 pm

https://developer.safaricom.co.ke

Configure Firebase

1. Fire up your Browser (I use chrome). Open firebase Console in https://firebase.google.com

2. Go to Console
3. Create new project
4. Click on **Firestore Database** to create a new database in **production mode.** We leave it there for now
5. Click Project **Overview/ Project settings**
6. Navigate to **Service accounts** tab and **Generate new private key**



Firebase Admin SDK

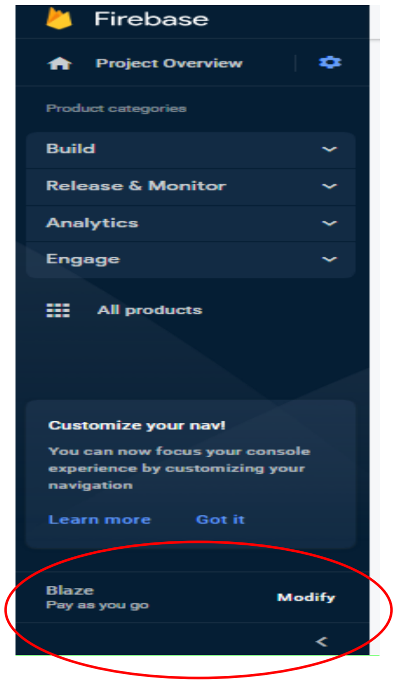
Your Firebase service account can be used to authenticate multiple Firebase features, such as Database, Storage and Auth, programmatically via the unified Admin SDK. Learn more

Firebase service account
firebase-adminsdk-3wzgl@mpesa-node-71fc5.iam.gserviceaccount.com

Admin SDK configuration snippet: [✓ Node.js] Java Python Go

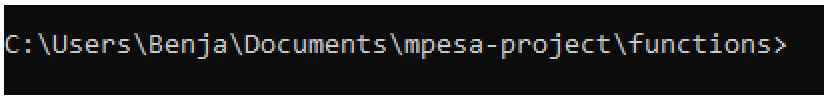
```
var admin = require("firebase-admin");
```

7.
8. Private key downloads automatically, Open to view downloaded json file then copy all code content there and paste it in your projects **config.json** file
9. Go back to console and upgrade your project from **Spark plan** to **Blaze plan.** Without this step you will be unable to deploy firebase functions. You will also require a valid Credit/Debit card for this step. (Do not worry. They offer free function invocation for about 2 million requests per month. That's generous)



Upload project to Firebase Functions

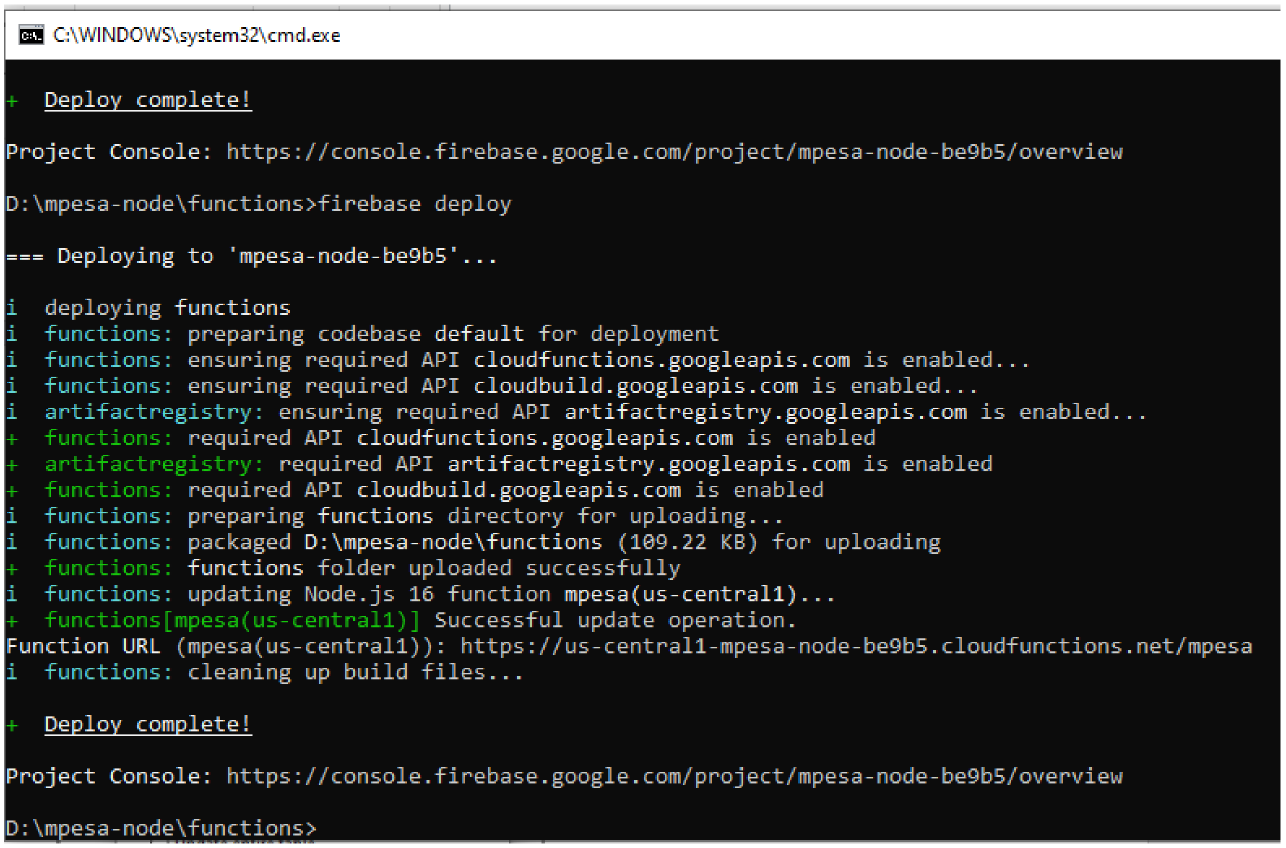
1. Make sure node and npm is installed into your machine
2. After successfully upgrading your firebase project, fire up command prompt (cmd)
3. Install firebase sdk tools to your machine using npm e.g **npm install -g firebase-tools**
4. Exit cmd then fire it once again
5. Cd (change directory) in the command prompt to the root path where you project is located. E.g **cd C:\Users\Benja\Documents\mpesa-project**
6. Fire up google chrome and leave it running, then in the cmd run **firebase login**
7. After successful login, run **firebase init functions** to initialize your firebase functions
8. Proceed with **yes**, overwrite **codebase** (if they prompt), select **javascript** for language, select **no** for **eslint**, select **no** for **package.json**, select **no** for **index.js**, select **no** for **.gitignore**, select **yes** for **install dependencies with npm**
9. Cd to the functions folder of our project e.g **C:\Users\Benja\Documents\mpesa-project\functions**



```
C:\Users\Benja\Documents\mpesa-project\functions>
```

10. Run **firebase deploy**

11. After successful deploy, head over to your firebase console, then click on functions. Copy the url where your functions have deployed upon. Paste that url into **FUNCTIONS_URL** in **mpesa-config.json**



C:\WINDOWS\system32\cmd.exe

```
+  Deploy complete!

Project Console: https://console.firebase.google.com/project/mpesa-node-be9b5/overview

D:\mpesa-node\functions>firebase deploy

=== Deploying to 'mpesa-node-be9b5'...

i  deploying functions
i  functions: preparing codebase default for deployment
i  functions: ensuring required API cloudfunctions.googleapis.com is enabled...
i  functions: ensuring required API cloudbuild.googleapis.com is enabled...
i  artifactregistry: ensuring required API artifactregistry.googleapis.com is enabled...
+  functions: required API cloudfunctions.googleapis.com is enabled
+  artifactregistry: required API artifactregistry.googleapis.com is enabled
+  functions: required API cloudbuild.googleapis.com is enabled
i  functions: preparing functions directory for uploading...
i  functions: packaged D:\mpesa-node\functions (109.22 KB) for uploading
+  functions: functions folder uploaded successfully
i  functions: updating Node.js 16 function mpesa(us-central1)...
+  functions[mpesa(us-central1)] Successful update operation.
Function URL (mpesa(us-central1)): https://us-central1-mpesa-node-be9b5.cloudfunctions.net/mpesa
i  functions: cleaning up build files...

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/mpesa-node-be9b5/overview

D:\mpesa-node\functions>
```

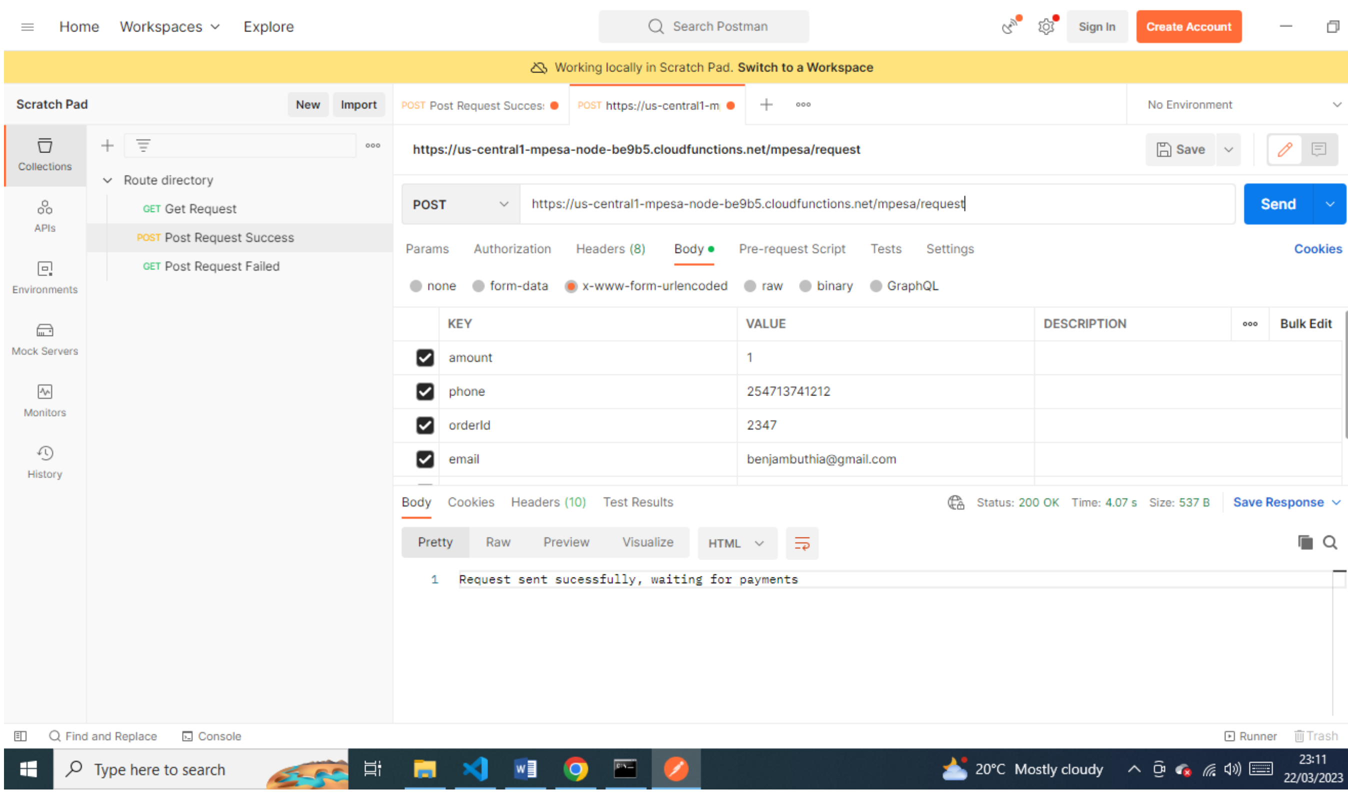
Configure woocommerce

1. To configure woocommerce, head over to woocommerce-config.json replace **url**, **consumerKey**, **consumerSecret**. Get this credentials from your wordpress

N.B

*Note that you when you create an order in your mobile app, set key-value' s (**status** to **pending** and **set_paid** to **false**), this script only updates the order status after successful payments. After successful payments, this script set (**status** to **completed** and set_**paid** to **true**)*

Test in Postman



https://us-central1-mpesa-node-be9b5.cloudfunctions.net/mpesa/request

POST https://us-central1-mpesa-node-be9b5.cloudfunctions.net/mpesa/request

Body: x-www-form-urlencoded

KEY	VALUE	DESCRIPTION
amount	1	
phone	254713741212	
orderId	2347	
email	benjambuthia@gmail.com	

Status: 200 OK Time: 4.07 s Size: 537 B

```
Request sent sucessfully, waiting for payments
```

```
Use POST http verb. Paste in the url where functions deployed. Pass
[amount, phone, orderId, email, customerId ]
```

Going Live

Once Safaricom mpesa approvs you for live and sends you the live passkey. Update live passkey, live ConsumerKey and live ConsumerSecret values in your mpesa-config.json file

You will also need to update this credentials in your firebase functions.

In the mpesa-utils.js , change line

```
const shortcode= config.SANDBOX_BUSINESS_SHORTCODE;
const passkey= config.SANDBOX_PASSKEY;
```

to

```
const shortcode= config.LIVE_BUSINESS_SHORTCODE;
const passkey= config.LIVE_PASSKEY;
```

also

```
In the index.js change line
const auth_token_url= config.SANDBOX_AUTH_URL;
const stkpush_url= config.SANDBOX_STKPUSH_URL;
const consumer_key= config.SANDBOX_CONSUMER_KEY;
const consumer_secret= config.SANDBOX_CONSUMER_SECRET;
```

to

```
In the index.js change line
const auth_token_url= config.LIVE_AUTH_URL;
const stkpush_url= config.LIVE_STKPUSH_URL;
const consumer_key= config.LIVE_CONSUMER_KEY;
const consumer_secret= config.LIVE_CONSUMER_SECRET;
```

Usage in Android (Kotlin)

```
// First, create a data class to represent the parameters
data class MpesaRequest(
    val amount: String,
    val phone: String,
    val orderId: String,
    val email: String,
    val customerId: String
)

// Then, create a Retrofit interface for the API endpoint
interface MpesaService {
    @POST("mpesa")
    suspend fun postMpesaRequest(@Body request: MpesaRequest): Response<ResponseBody>
}

// Finally, make the network call using Retrofit and Coroutine to post the data
val retrofit = Retrofit.Builder()
    .baseUrl("https://us-central1-mpesa-node-be9b5.cloudfunctions.net/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

val mpesaService = retrofit.create(MpesaService::class.java)

// Create the MpesaRequest object with the parameters
val request = MpesaRequest(
    amount = "100",
    phone = "254712345678",
    orderId = "123456789",
    email = "test@example.com",
```

Note: This example uses the kotlinx-coroutines-core and retrofit2 libraries. Don't forget to add them to your project dependencies.

Usage in Android (Java)

```
// First, create a POJO class to represent the parameters
public class MpesaRequest {
    private String amount;
    private String phone;
    private String orderId;
    private String email;
    private String customerId;

    public MpesaRequest(String amount, String phone, String orderId, String email, String customerId) {
        this.amount = amount;
        this.phone = phone;
        this.orderId = orderId;
        this.email = email;
        this.customerId = customerId;
    }

    // getters and setters here
    // ...
}

// Then, create a Retrofit interface for the API endpoint
public interface MpesaService {
    @POST("mpesa")
    Call<ResponseBody> postMpesaRequest(@Body MpesaRequest request);
}

// Finally, make the network call using Retrofit to post the data
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://us-central1-mpesa-node-be9b5.cloudfunctions.net/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();
```

Usage in IOS

```
import Alamofire


let parameters: Parameters = [
   "amount": "100",
   "phone": "254712345678",
   "orderId": "12345",
   "email": "example@example.com",
   "customerId": "67890"
]


let url = "https://us-central1-mpesa-node-be9b5.cloudfunctions.net/mpesa"


AF.request(url, method: .post, parameters: parameters, encoding: JSONEncoding.default)
   .responseJSON { response in
      switch response.result {
      case .success(let value):
         print("Response: \(value)")
      case .failure(let error):
         print("Error: \(error)")
      }
   }
```

In this example, we create a dictionary with the parameters to be passed in the request, and then use the `AF.request` method to make the request. We specify the URL and HTTP method (in this case, `POST`), and pass in the parameters using JSON encoding. We then handle the response using a closure that prints the response data if the request was successful, or the error if it failed.

Make sure to include the Alamofire library in your project before using it in your code.

Usage in react-native

In this code, the `postData` function makes a POST request to the specified URL with the provided parameters in the `data` object. The `requestOptions` variable is an object that contains the configuration for the POST request, including the `Content-Type` header set to `application/json` and the request body as a JSON string of the `data` object. The `fetch` function is then used to make the actual HTTP request. Finally, the response is parsed as JSON and logged to the console.

Note that you may need to modify this code to fit the specific requirements of your project, such as replacing the URL or adjusting the data parameters.

```
const postData = async () => {
 const url = 'https://us-central1-mpesa-node-be9b5.cloudfunctions.net/mpesa';
 const amount = 100;
 const phone = '254712345678';
 const orderId = '1234';
 const email = 'example@example.com';
 const customerId = '5678';

 const data = {
  amount,
  phone,
  orderId,
  email,
  customerId,
 };

 const requestOptions = {
  method: 'POST',
  headers: {
   'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
 };
 try {
  const response = await fetch(url, requestOptions);
  const responseData = await response.json();
  console.log(responseData);
 } catch (error) {
  console.error(error);
 }
}
```