ChatGPT

Rwanda MTN MoMo and Stripe Payment Integration

In Rwanda, MTN Rwandacell's Mobile Money (MoMo) service is a dominant way to send and receive payments. MTN provides a public **MoMo API** that lets websites request payments from customers' mobile wallets. Using this API, an online store can initiate a "request to pay" prompt on a customer's phone, who then approves the charge with their PIN [1]. Developers can integrate this API using Node.js (many SDKs exist) to handle the server-side calls. Meanwhile, for customers in Europe or elsewhere, the site can use Stripe's payment platform (card payments) alongside MoMo. By combining MoMo and Stripe, a site offers regional payment options: African users pay with MoMo, and European users pay by card.

Obtaining MTN MoMo API Access

To use the MTN MoMo API, a developer must register on MTN's developer portal and obtain API credentials. This involves signing up on the *MoMo Developer Portal* and requesting access to the MoMo Collection API. MTN then issues a **subscription key**, an **API user ID**, and an **API key** for your application. (An example guide notes that an active MTN MoMo API subscription key is required to proceed [2].) In sandbox mode (for testing), MTN provides dummy endpoints and allows you to create API users programmatically. In production, you must set these values to the ones MTN gives you. All MoMo calls (sandbox or live) require the subscription key and bearer token for authentication.

MTN MoMo API Workflow

The MoMo Collection API (used for charging customers) follows these steps:
- **Create API User (Sandbox only):**
Send `POST /v1_0/apiuser` with a unique reference. This registers a new API user in the sandbox environment [3].
- **Generate API Key:**
Send `POST /v1_0/apiuser/{userId}/apikey` (using the reference ID) to obtain a secret API key [3].
- **Obtain Access Token:**
Call `POST /collection/token` with Basic Auth (using `api_user_id:api_key`) and the `Ocp-Apim-Subscription-Key` header. The response is a Bearer token for authorization [3].
- **Request To Pay:**
Call `POST /collection/v1_0/requesttopay` with a JSON body specifying the **amount**, **currency**, **externalId** (your own reference), and **payer** (their MSISDN/phone). Include headers: `X-Reference-Id` (unique request ID), `X-Target-Environment` (e.g. "sandbox"), and the subscription key and Bearer token. This initiates the payment on the customer's phone [1].
- **Check Payment Status:**
Optionally, call `GET /collection/v1_0/requesttopay/{referenceId}` to poll the payment status (success, pending, failed) [3]. If you've configured a callback URL, MTN will POST the result to your endpoint when the payment completes.

This workflow is documented in MTN's API guide and Node SDKs. All requests must include your **subscription key** and the properly scoped Bearer token. The MoMo API uses local currency (for Rwanda, RWF) and operates in real-time.

Integrating MoMo in Node.js

In a Node.js codebase, you can call the MTN MoMo API directly using HTTP or use an SDK like `mtn-momo-sdk`. For example:
- **Install SDK:**

```
npm install mtn-momo-sdk
```

The package's README notes it supports both sandbox and production modes, handles token generation, and even auto-provisions API users in sandbox [4] [5].
- **Configure Credentials:**
Create a `.env` file with your MTN_MOMO environment. For sandbox you might set:

```
MTN_MOMO_ENV=sandbox
MOMO_API_BASE_URL_SANDBOX=https://sandbox.momodeveloper.mtn.com
X_TARGET_ENVIRONMENT_SANDBOX=sandbox
SUBSCRIPTION_KEY_SANDBOX=your_sandbox_subscription_key_here
PROVIDER_CALLBACK_HOST_SANDBOX=https://your-callback-url.com
```

In production mode you would set `MOMO_API_BASE_URL_PRODUCTION`, `SUBSCRIPTION_KEY_PRODUCTION`, `API_USER_PRODUCTION`, and `API_KEY_PRODUCTION` instead [6]. These variables ensure the SDK (or your code) knows which keys and endpoints to use.
- **Making a Payment Request:**
Once configured, call the SDK's function. For example, in code:

```
const { processPayment } = require('mtn-momo-sdk');
let result = await processPayment('100.00', '2507XXXXXXX', 'Invoice #123');
console.log(result.status);  // e.g. 'SUCCESSFUL' or 'FAILED'
```

Here `processPayment` handles creating the token, sending the request-to-pay, and polling for the result [7]. Without the SDK, you'd manually replicate these steps: generate token, POST to `/requesttopay`, etc.

MoMo Payment Flow

Using the SDK or your HTTP calls, the payment flow is as follows:
1. **Environment Setup:** The code checks if it's in sandbox or production (based on `MTN_MOMO_ENV`). In sandbox, the SDK can automatically create an API user and key for you per transaction; in production it uses the pre-configured ones [8].
2. **Token Generation:** The code POSTs to `/collection/token` with your API user/key to get a Bearer token.

3. **Initiate Charge:** The code sends the request-to-pay with amount, payer number, and reference. At this point, MTN's system will send a prompt to the user's phone asking them to approve the payment.
4. **Confirmation:** If you provided a callback URL (configured in `PROVIDER_CALLBACK_HOST`), MTN will POST the transaction result to your server. Otherwise, your Node code should poll `/requesttopay/{ref}` until it sees success or timeout. The `mtn-momo-sdk` even handles fallback polling if callbacks fail [9].
5. **Result Handling:** Finally, your code receives a status (e.g. `SUCCESSFUL` or `FAILED`) and can mark the order accordingly. The SDK provides detailed error messages if something goes wrong [3].

Throughout, make sure to handle errors and implement retries or timeouts. The Node SDK's docs note it will retry failed requests and log full API responses [10].

Meanwhile, for European or international customers, you can use Stripe's payment platform to accept cards and other methods. Stripe offers a well-documented Node.js library and frontend elements for payments. To integrate Stripe:
- **Install Stripe:** Include Stripe's Node SDK in your project (`npm install stripe`). Initialize it in code with your secret key:

```
const stripe = require('stripe')('sk_test_…');
```

- **Create a Payment Intent:** On your server, when the customer checks out, call Stripe's API. For example:

```
const paymentIntent = await stripe.paymentIntents.create({
  amount: 1099,
  currency: 'eur',
  // optionally attach a customer or payment method, etc.
  automatic_payment_methods: {enabled: true},
});
```

This returns a `client_secret` which you send to the frontend. On the client side, you use Stripe.js or Stripe Elements to confirm the payment (enter card info, 3D Secure, etc.). This flow is described in Stripe's documentation [11]. Stripe will then process the card, and you can listen for its payment succeeded event (via webhook or callback) to finalize the order.

Combining MoMo and Stripe (Checkout Flow)

- **Customer Chooses Method:** On the checkout page, present options: e.g. **Mobile Money (MTN MoMo)** or **Credit/Debit Card (Stripe)**. This lets the user pick the payment relevant to their region.
- **Mobile Money Flow:** If the user selects MTN MoMo, collect their phone number (including country code, e.g. +2507XXXXXXXX). Send this info to your backend endpoint (e.g. `/api/pay/momo`). The server then calls the MoMo API as described above (using `processPayment` or equivalent). Inform the user to watch their phone and enter their MoMo PIN to approve. The backend waits for success via callback or polling, then confirms payment on the site.
- **Stripe Flow:** If the user chooses Stripe, present a secure card input (Stripe Elements or Checkout). On submit, create a PaymentIntent on the server with the order amount in EUR (or desired currency)

[11]. Return the `client_secret` to the client and confirm it with Stripe.js. When Stripe reports success, mark the order as paid.

- **Currency and Country Handling:** Note that MTN MoMo charges are in local currency (Rwandan Francs, RWF), whereas Stripe might use EUR or USD. Ensure you specify the correct currency in each call. You may also tailor the UI (e.g. show MoMo only to users in Rwanda/Uganda where MTN operates, and Stripe elsewhere). All secret keys (MoMo subscription key, Stripe API keys) should be stored securely on your server, not exposed to the client.

Summary

By integrating both systems, your website can seamlessly serve different regions. Node.js is a common backend for this: one part of your code handles MoMo API calls (using packages like **mtn-momo-sdk** [4] [7]), and another handles Stripe calls. Users simply pick their payment method ("customer can choose"), and the server routes to the appropriate API. This hybrid setup enables payment collection across Africa (via MTN MoMo) and Europe (via Stripe) in one codebase. All technical details — obtaining API credentials, configuring environments, and handling callbacks — are documented in the official guides and SDKs [6] [11], ensuring a robust, fully featured integration.

Sources: MTN MoMo Developer docs and SDK examples [3] [1] [6]; Stripe API documentation and tutorials [11] [4].

---

[1] [2] Building a Payment Gateway using MTN Mobile Money(MOMO) API in Python | by George S. Mulbah | Medium
https://medium.com/@gsmulbah2500/building-a-payment-gateway-using-mtn-mobile-money-api-in-python-939315f0f08a

[3] [4] [5] [6] [7] [8] [9] [10] GitHub - DamianoSilverhand/mtn-momo-node: This repository hosts a Node.js SDK for integrating with MTN MoMo (Mobile Money) API, specifically tailored for payment collections
https://github.com/DamianoSilverhand/mtn-momo-node

[11] docs.stripe.com
https://docs.stripe.com/payments/quickstart?client=react&lang=node