# Table of Contents

**Versions:**

| Version | Date | Changes | Author |
|---|---|---|---|
| 1.0.0 | October 2024 | Initial documentation for M-Pesa and Card Payment services | Cashia |
| 1.0.1 | November 2024 | Card 3DS transaction details added | Cashia |
| 1.0.2 | November 2024 | Transaction fee calculation service added, fee-amount and payment method code added to card payment and Mpesa payment service, business-id from the card payment services removed | Cashia |
| 1.0.3 | November 2024 | Payout Bank Accounts service added | Cashia |
| 1.0.4 | December 2024 | Uniformed the inputs and outputs: OrderId -> RequestId, PayoutAmount -> Amount, Payment Mpesa Refund inputs updated, Error message structure updated | Cashia |
| 1.0.5 | December 2024 | Hosted checkout instructions added | Cashia |
| 1.0.6 | December 2024 | fee-amount is removed from payment services. PGW calculates the customer amount and adds it to the transaction amount. The response of Get Payment Method extended | Cashia |
| 1.0.7 | February 2025 | Payout to M-Pesa Phone Number, Payout to M-Pesa Paybill, Payout to M-Pesa Buy Goods Till Number services added | Cashia |
| 1.0.8 | May 2025 | Revamped API scheme for new gateway environment | Cashia |
| 1.0.9 | June 2025 | Updated Authentication approach | Cashia |

# Integration - API References

Cashia's API is built on REST principles. It features clear, resource-based URLs, processes form-encoded request bodies, returns responses in JSON format, and follows standard HTTP methods, response codes, and authentication practices.

You can run the API in test mode, allowing you to simulate requests without impacting real data or financial networks. The API key you use will determine whether your requests are processed in live or test mode.

Cashia supports Host-to-host connection API, and hosted checkout page, providing a flexible way to integrate directly with our platform.

Cashia offers seamless integration through RESTful API calls, with support for payment status notifications via callbacks. To accept payments from your customers, you can either use our pre-built Checkout page or create a custom payment page tailored to your needs.

### Callbacks

Cashia’s callbacks keep you informed about payment-related events, such as changes in the status of transactions or payouts. These notifications ensure that you stay updated in real-time, even after the customer has left your site, allowing for timely responses to any developments. Callback URLs are defined on business definition.

### Domains and IPs

For a secure and smooth integration, it’s essential to communicate with {CASHIA API URL} over a connection secured by an SSL certificate.

To further enhance the security of interactions between the Cashia platform and your server, ensure that you use a whitelist of trusted IP addresses.

### Whitelist IP Addresses

To enhance the security of interactions between the Cashia platform and your server, it is recommended to use a whitelist of trusted IP addresses. This ensures that only authorized IPs can communicate with your server for callbacks and payment reconciliation.

**Complete List of IP Addresses Used by Cashia for Callbacks and Payment Reconciliation:**

- IP ADDRESS 1 - to be shared
- IP ADDRESS 2 - to be shared

By whitelisting these IP addresses, you ensure that your server only accepts connections from the Cashia platform, providing an additional layer of security.

### Authentication

### API Access

The Cashia API follows REST architecture, offering resource-based URLs. It supports form-encoded request bodies, returns responses in JSON format, and uses standard HTTP methods, authentication, and response codes.

Integration API requests use either live or test Public Keys and Secrets, which Cashia will provide you before the Integration starts.

### Overview

This API uses HMAC SHA256 for authentication. Each request must include an authentication header containing a key id, nonce, timestamp, body hash, and URL signature. The shared secret between the client and server will be used to generate the HMAC signature.

### Authentication Header Structure

The authentication header should contain the following fields:

- **Authorization**: A string containing the authentication tag and the parameters.

**Required Parameters**

**Key ID:** The public key shared along with the private key

**Nonce**: A unique string is generated for each request. It prevents replay attacks. Minimum 12 length.

**Timestamp**: The current Unix time for the UTC.

**Body Hash**: A HMAC SHA256 hash of the request body.

**Signature**: A HMAC SHA256 URL hash, using the shared secret. Formated base64.

**Header Example**

```
None
X-Cashia-Key-ID: 01927a78-d9af-75ab-a849-25a2b8f67be5
X-Cashia-Nonce: unique_nonce
X-Cashia-Timestamp: 1729220568
X-Cashia-Hash: 478131ca50c3a2e7f14e782abcfa04a91e45491f15b5
X-Cashia-Signature: 478131ca50c3a2e7f14e782abcfa04a91e45491f15b5
```

**Generate Authentication Step By Step**

- **Shared Secret**: `my_secret_key`
- **Key ID:** `01927a78-d9af-75ab-a849-25a2b8f67be5`
- **Nonce**: `randomstring`
- **Timestamp**: `1729220568`
- **HTTP Method**: `POST`
- **URL**: `https://api.example.com/resource`
- **Body**: `{"example":"data"}`

1. **Generate Body Hash**

**Body**: `{"example":"data"}`
**HMAC SHA256 Hash:**

```
None
d43e8dfa81aa999389ff52a2a93cc50d0c186545e38e337d44ed95ecedabaf06
```

2. **Create the Signing String**:

   **String**: Combine the host, HTTP method, timestamp, nonce, and key id:

```
None
api.example.com POST1729220568randomstring01927a78-d9af-75ab-a849-25a2b8f67be5
```

3. **Compute Signature**:

   **HMAC SHA256 Signature**:

   - Use the shared secret (`my_secret_key`) and signing string to compute:

```
None
516ec804fa3be38d8e2d0a087e12d1e17f59d0d53823b095f75d72b301d8a3dd
```

4. **Construct the Authorization Header**:

   Final Header

```
None
'X-Cashia-Key-ID': '01927a78-d9af-75ab-a849-25a2b8f67be5',
'X-Cashia-Timestamp': '1729220568',
'X-Cashia-Signature':
'516ec804fa3be38d8e2d0a087e12d1e17f59d0d53823b095f75d72b301d8a3dd',
'X-Cashia-Nonce': 'randomstring',
'X-Cashia-Hash': 'd43e8dfa81aa999389ff52a2a93cc50d0c186545e38e337d44ed95ecedabaf06'
```

### Example Code in Nodejs

Here's a simple SDK in NodeJs that illustrates how to generate the authentication header.

```javascript
const request = require('request');
const crypto = require('crypto');


function computeHmac256(message, secret) {
  const hmac = crypto.createHmac('sha256', secret);
  hmac.update(message);
  return hmac.digest('hex');
}

const secret = 'noSHBWj94PL8FXkSAdE4h+nR09iQZ5n8';
const host = 'localhost:8000';
const timestamp = Math.floor(Date.now() / 1000);
const keyID = '01927a78-d9af-75ab-a849-25a2b8f67be5';
const nonce = 'nonce';
const signatureRaw = `${host}POST${timestamp}${nonce}${keyID}`;

const data = JSON.stringify({
  "age": 16,
  "email": "user@gmail.com"
});

const headers = {
    'X-Cashia-Key-ID': keyID,
    'X-Cashia-Timestamp': timestamp,
    'X-Cashia-Signature': computeHmac256(signatureRaw, secret),
    'X-Cashia-Nonce': nonce,
    'X-Cashia-Hash': computeHmac256(data, secret),
};

const options = {
  'method': 'POST',
  'url': `http://${host}/api/v1/merchant-info`,
  'headers': headers,
```

```
  body: data
};

request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

### Example Code in Go

Here's a simple SDK in Go that illustrates how to generate the authentication header.

```C/C++
package main

import (
	"bytes"
	"crypto/hmac"
	"crypto/sha256"
	"encoding/hex"
	"encoding/json"
	"fmt"
	"io"
	"net/http"
	"strconv"
	"time"
)

func computeHmac256(message, secret string) string {
	h := hmac.New(sha256.New, []byte(secret))
	h.Write([]byte(message))
	return hex.EncodeToString(h.Sum(nil))
}

func main() {
	secret := "noSHBWj94PL8FXkSAdE4h+nR09iQZ5n8"
	host := "localhost:8000"
```

```
	timestamp := strconv.FormatInt(time.Now().Unix(), 10)
	keyID := "01927a78-d9af-75ab-a849-25a2b8f67be5"
	nonce := "nonce"
	signatureRaw := host + "POST" + timestamp + nonce + keyID

	data := map[string]interface{}{
		"age":   16,
		"email": "user@gmail.com",
	}

	jsonData, err := json.Marshal(data)
	if err != nil {
		panic(err)
	}

	headers := map[string]string{
		"X-Cashia-Key-ID":    keyID,
		"X-Cashia-Timestamp": timestamp,
		"X-Cashia-Signature": computeHmac256(signatureRaw, secret),
		"X-Cashia-Nonce":     nonce,
		"X-Cashia-Hash":      computeHmac256(string(jsonData), secret),
	}

	url := "http://" + host + "/api/v1/merchant-info"
	response := makeHttpRequest(url, "POST", headers, jsonData)
	fmt.Println(string(response))
}

func makeHttpRequest(url, method string, headers map[string]string, jsonData
[]byte) []byte {
	req, err := http.NewRequest(method, url, bytes.NewBuffer(jsonData))
	if err != nil {
		panic(err)
	}

	for key, value := range headers {
		req.Header.Set(key, value)
	}

	client := &http.Client{}
	resp, err := client.Do(req)
	if err != nil {
```

```
            panic(err)
        }
        defer resp.Body.Close()

        body, err := io.ReadAll(resp.Body)
        if err != nil {
            panic(err)
        }

        return body
}
```

### Integration API

For integration, use the following API endpoints based on your environment:

- **Test Environment:** http://pre-prod.cashia.com/api
- **Live Environment:** TBD

Ensure you switch between test and live environments as needed to manage your payment processing securely.

### Swagger Documentation

You can access the full list of available APIs by adding the following swagger url at the end of the test and live environments

Endpoints: swagger.cashia.com

# API References

## Transaction Status

The Transaction Status Inquiry service allows businesses to check the current status of a specific transaction. This service takes the transaction ID as input and returns detailed information about the transaction, including its status, amount, and currency.

- **API:** Public
- **Authentication:** Live or test Public Keys
- **Endpoint:** `/v1/transactions/:transactionId`
- **Method:** GET

### Request Object

The request object for inquiring about a transaction status contains the following field:

```
"transactionId": "string"   // Required: The Transaction ID
```

- transaction/33cb972f-9fd7-4ec7-b94b-9a3584965fc8

#### Request Fields

- **transactionId (string, required)**: The unique identifier of the transaction whose status you want to check.

#### Response Fields

#### Example Response (Success)

JSON

```
{
  "process": {
    "id": "string",
    "createdAt": "2025-05-30T12:56:05.805Z",
    "updatedAt": "2025-05-30T12:56:05.805Z",
    "type": "string",
```


```
"status": "limited",
"categoryCode": "string",
"categoryName": "string",
"categoryImageLink": "string",
"requestIdentifier": "string",
"requestStatus": "pending",
"internalSourceAmount": 0,
"internalDestinationAmount": 0,
"externalSourceAmount": 0,
"externalDestinationAmount": 0,
"externalProcessId": "string",
"underInvestigation": true,
"refundProcess": "string",
"txIdToRefund": "string",
"sourceCurrency": {
  "id": "string",
  "sn": "string",
  "code": "string",
  "symbol": "string",
  "type": "FIAT"
},
"destinationCurrency": {
  "id": "string",
  "sn": "string",
  "code": "string",
  "symbol": "string",
  "type": "FIAT"
},
"transactions": [
  {
    "id": "string",
    "parentId": "string",
    "type": "transfer",
    "from": {
      "serial": "string",
```

```
      "organizationId": "string",
      "organizationName": "string",
      "technical": true,
      "type": "client",
      "currency": {
        "id": "string",
        "sn": "string",
        "code": "string",
        "symbol": "string"
      },
      "name": "string",
      "accounting": true,
      "amount": 0,
      "heldAmount": 0
    },
    "to": {
      "serial": "string",
      "organizationId": "string",
      "organizationName": "string",
      "technical": true,
      "type": "client",
      "currency": {
        "id": "string",
        "sn": "string",
        "code": "string",
        "symbol": "string"
      },
      "name": "string",
      "accounting": true,
      "amount": 0,
      "heldAmount": 0
    },
    "amount": 0,
    "performedAt": "2025-05-30T12:56:05.805Z",
    "currency": {
```


```
        "id": "string",
        "sn": "string",
        "code": "string",
        "symbol": "string"
      }
    }
  ],
  "errorMessage": "string",
  "customInformation": "string",
  "commissionAmounts": {
    "source": {
      "total": 0,
      "system": 0,
      "provider": 0,
      "providerDebt": 0
    },
    "destination": {
      "total": 0,
      "system": 0,
      "provider": 0,
      "providerDebt": 0
    }
  },
  "commissionAmount": 0,
  "amount": 0,
  "description": "string",
  "coinDto": [
    {
      "serial": "string",
      "organizationId": "string",
      "organizationName": "string",
      "technical": true,
      "type": "client",
      "currency": {
        "id": "string",
```


```
          "sn": "string",
          "code": "string",
          "symbol": "string"
        },
        "name": "string",
        "accounting": true,
        "amount": 0,
        "heldAmount": 0
      }
    ],
    "note": "string",
    "initiatorFullName": "string"
  }
}
```


### Example Response (Failed Case)

Please see swagger, /v1/transactions/{transactionId}

## Merchant Wallet Balance

Retrieve the current balance details for a merchant's wallet.

- **API:** Public
- **Authentication:** Live or test Public Keys
- **Endpoint:** `/v1/coins`
- **Method:** GET

### Example Request

/v1/coins

### Example Response

The current balance details for a merchant's wallet, including available and hold balance. See example

None


```
{
    "coins": [
        {
            "serial": "527301835563",
            "name": "Test GBP coin",
            "amount": 10000.0000,
            "availableAmount": 10000.0000,
            "futureAmount": 0.0000,
            "heldAmount": 0.0000,
            "creditLimit": 0.0000,
            "currency": {
                "id": "01967113-89c9-76bc-b03a-e93980d2ce1a",
                "sn": "GBP",
                "code": "GBP",
                "symbol": "£",
                "type": "FIAT"
            },
            "active": true,
            "type": "client",
            "main": true,
            "accounting": false,
            "smartCards": []
        },
        {
            "serial": "383573772234",
            "name": "Test EUR coin",
            "amount": 10000.0000,
            "availableAmount": 10000.0000,
            "futureAmount": 0.0000,
            "heldAmount": 0.0000,
            "creditLimit": 0.0000,
            "currency": {
                "id": "01967113-8998-7a71-9a26-512389733e67",
                "sn": "EUR",
                "code": "EUR",
                "symbol": "€",
```

```
            "type": "FIAT"
        },
        "active": true,
        "type": "client",
        "main": true,
        "accounting": false,
        "smartCards": []
    },
    {
        "serial": "112071864149",
        "name": "Test USD coin",
        "amount": 10000.0000,
        "availableAmount": 10000.0000,
        "futureAmount": 0.0000,
        "heldAmount": 0.0000,
        "creditLimit": 0.0000,
        "currency": {
            "id": "01967113-8969-7124-aebb-efb30f0a143e",
            "sn": "USD",
            "code": "USD",
            "symbol": "$",
            "type": "FIAT"
        },
        "active": true,
        "type": "client",
        "main": true,
        "accounting": false,
        "smartCards": []
    },
    {
        "serial": "792488326835",
        "name": "Test USD coin",
        "amount": 0.0000,
        "availableAmount": 0.0000,
        "futureAmount": 0.0000,
        "heldAmount": 0.0000,
        "creditLimit": 0.0000,
        "currency": {
            "id": "01967113-8969-7124-aebb-efb30f0a143e",
            "sn": "USD",
            "code": "USD",
            "symbol": "$",
```

```
            "type": "FIAT"
        },
        "active": true,
        "type": "cash",
        "main": true,
        "accounting": false,
        "smartCards": []
    },
    {
        "serial": "555521637973",
        "name": "Test EUR coin",
        "amount": 0.0000,
        "availableAmount": 0.0000,
        "futureAmount": 0.0000,
        "heldAmount": 0.0000,
        "creditLimit": 0.0000,
        "currency": {
            "id": "01967113-8998-7a71-9a26-512389733e67",
            "sn": "EUR",
            "code": "EUR",
            "symbol": "€",
            "type": "FIAT"
        },
        "active": true,
        "type": "cash",
        "main": true,
        "accounting": false,
        "smartCards": []
    },
    {
        "serial": "668518666366",
        "name": "Test GBP coin",
        "amount": 0.0000,
        "availableAmount": 0.0000,
        "futureAmount": 0.0000,
        "heldAmount": 0.0000,
        "creditLimit": 0.0000,
        "currency": {
            "id": "01967113-89c9-76bc-b03a-e93980d2ce1a",
            "sn": "GBP",
            "code": "GBP",
            "symbol": "£",
```


```
                "type": "FIAT"
            },
            "active": true,
            "type": "cash",
            "main": true,
            "accounting": false,
            "smartCards": []
        },
        {
            "serial": "399555936272",
            "name": "My Checkout Wallet",
            "amount": 0.000,
            "availableAmount": 0.000,
            "futureAmount": 0.000,
            "heldAmount": 0.000,
            "creditLimit": 0.000,
            "currency": {
                "id": "0196ee49-9995-74db-9c30-9b16586a2b9d",
                "sn": "KES",
                "code": "KES",
                "symbol": "/=",
                "type": "FIAT"
            },
            "active": true,
            "type": "client",
            "main": true,
            "accounting": false,
            "smartCards": []
        }
    ]
}
```


## HOSTED CHECKOUT

There are currently two ways to implement  the hosted checkout solution, using

1. Iframe
2. Redirection to payment page

The two approaches are similar and follow the following similar steps:

## Step 1: Create Checkout Session

Merchants will create the session for the payment to take place.

- **API:** Private
- **Authentication:** Live or Test Public Keys
- **Endpoint:** `/v1/hosted-checkout`
- **Method:** POST

Method: POST

### Example Request

```
{
 "requestId": "order2-1277",
 "currency": "USD",
 "amount": 151,
 "webhookUrl": "https://platypus-settling-likely.ngrok-free.app/callback",
 "successRedirectUrl": "https://example.com/success",
 "errorRedirectUrl": "https://example.com/error",
 "orderDetails": [
   {
     "name": "Shoes",
     "currency": "string",
     "quantity": 1,
     "description": "string",
     "price": 0
   }
 ],
 "deliveryDetails": {
   "currency": "USD",
   "fee": 0
 }
}
```

### Example Response

```
{
   "sessionId": "0193cebd-772f-713a-af5f-cf4f2c8591bf",
```

```
  "url":
"http://checkout.stg.cashia.com/link/MerchantName/0193cebd-772f-713a-af5f-cf4f2c8591bf",
  "amount": 10,
  "currency": "KES",
  "requestId": "order-1234"
}
```

### Step 2: Use the URL in your Code.

#### Iframe Approach

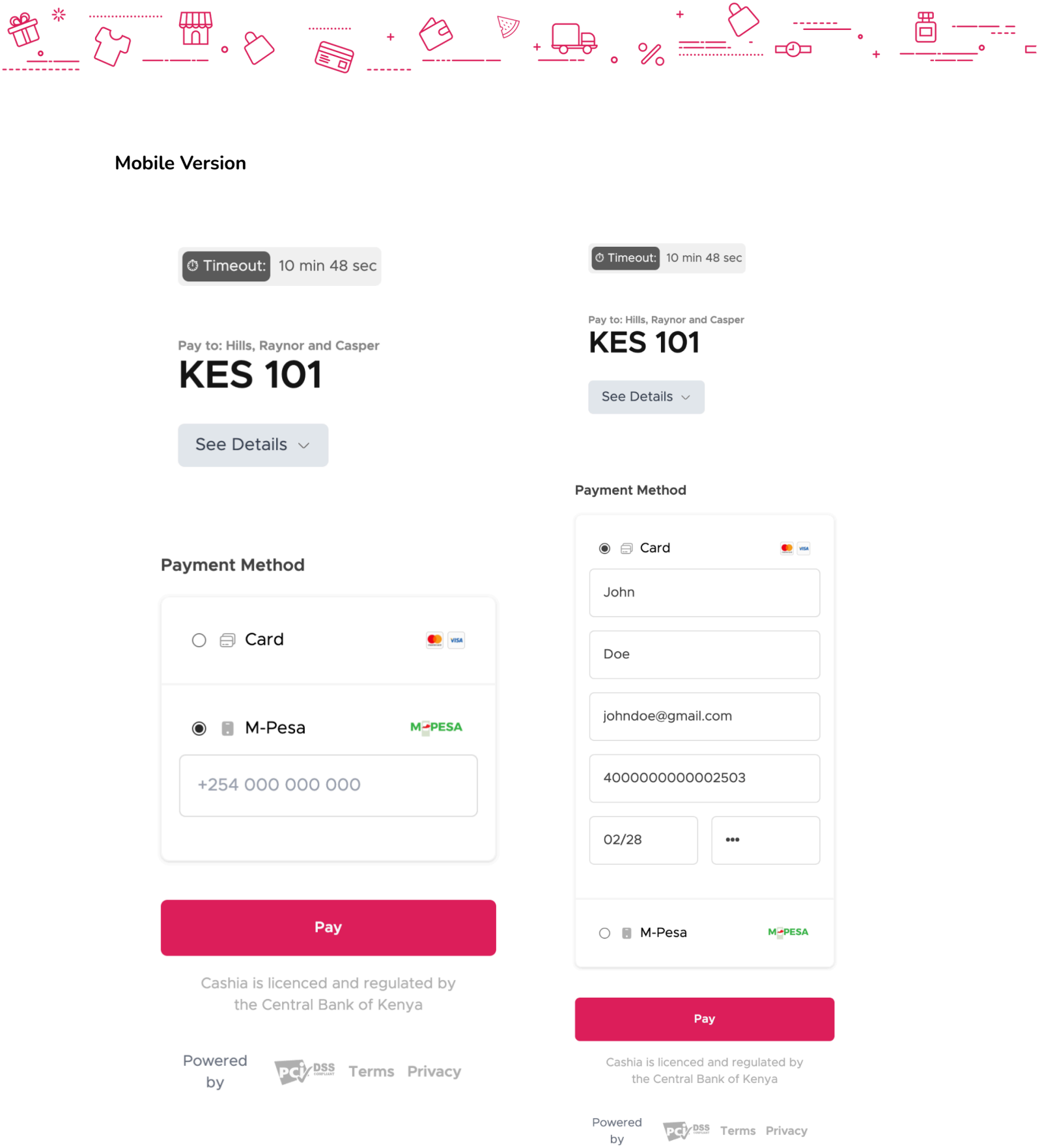
With the URL from the above response, you can now render an iframe as follows

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <title>Merchant Checkout Page</title>
</head>
<body>
   <div> Merchant Checkout Page</div>
   <iframe name="cashia" width="400" height="600"
src="https://checkout.stg.cashia.com/link/MerchantName/0193cee1-3206-7e
18-b981-37e036b4545f"></iframe>
</body>
</html>
```

#### Hosted Page Redirect Approach

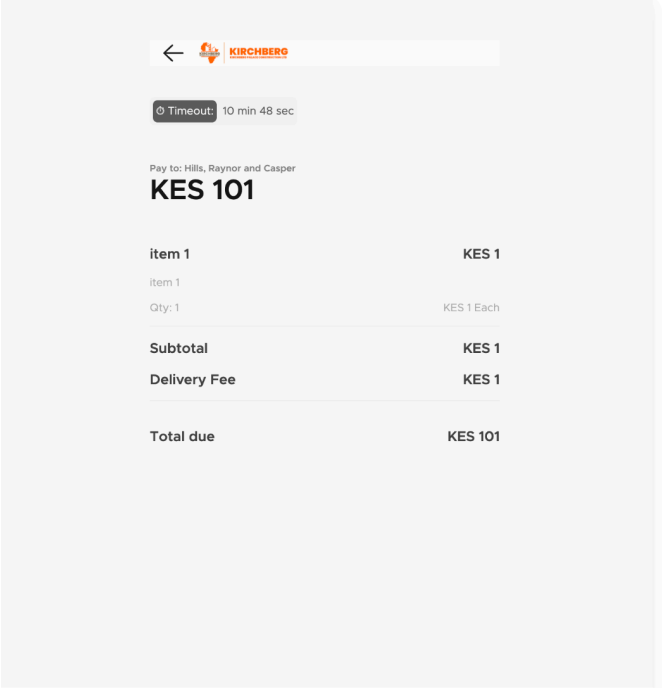
With the URL from the above response, you can redirect the user to the url where they can make the payment. The checkout page will appear as below.

## Mobile Version

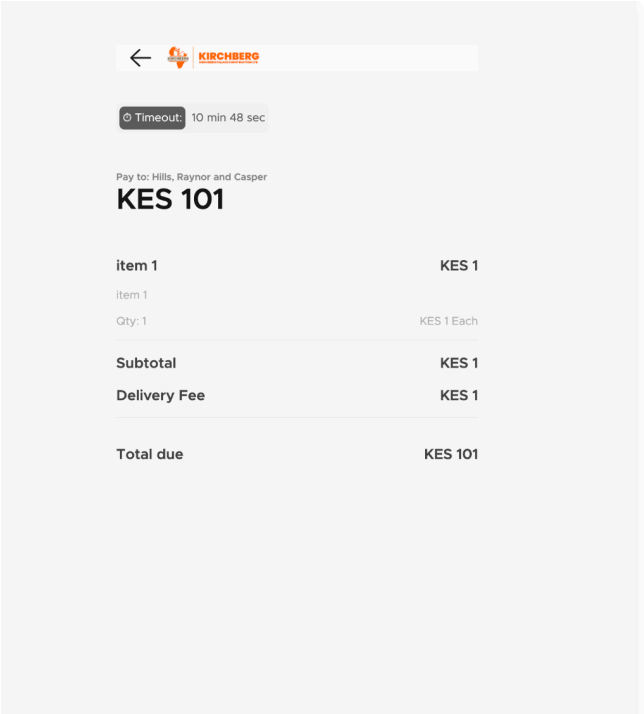


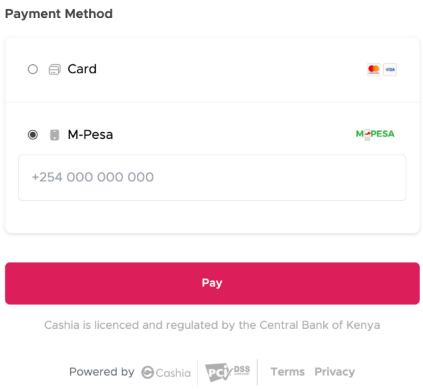
## On Desktop

## Card  Example




## MPESA Example




Here is a code example demonstrating how to redirect the user to the checkout page using a link.

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
   <title>Merchant Checkout Page</title>
 </head>
 <body>
   <div>Merchant Checkout Page</div>
   <div style="margin-bottom: 20px"></div>
   <a
href="https://checkout.stg.cashia.com/link/EmpireFX/0193cee1-3206-7e18-
b981-37e036b4545f" target="_blank"
   >
     <div>Click to Pay</div>
   </a>
 </body>
</html>
```

You will see a payment button similar to the one shown below.

### Step 3: Redirection to the merchant success/error page

Payment processing results in either a success or failure/cancellation. Upon completion, the end user must be notified of the payment outcome. successCallbackURL or errorCallbackURL is used for user redirection.

During session creation, the merchant is required to provide callback URLs for both success and failure scenarios. Based on the payment result, the payment gateway will trigger the corresponding callback URL.

To retrieve checkout details such as `orderId`, `session` details, and more, the merchant can include these as parameters in the session creation request. Below is an example of a success callback URL usage:

- "successCallbackURL": "https://example.com/success?validTimeStamp=1708118400"

Payment gateway adds the `requestId`, and `status` at the end of the request. The callback URL with additional information will be as:

- "https://example.com/success?validTimeStamp=1708118400&requestId=order2-1237&status=C"

## HTTP Status Codes Used in Responses

If the request fails, you will receive an error response with an appropriate HTTP status code and message. The following table lists the HTTP status codes returned by the Cashia platform based on different scenarios:

| Status Code | Description |
|---|---|
| **2xx Success** | |
| 200 OK | The request was successful. |
| 201 Created | A new resource was successfully created. |
| **4xx Client Errors** | |
| 400 Bad Request | The request was invalid or cannot be processed. |
| 401 Unauthorized | Authentication failed or not provided. |
| 403 Forbidden | Access to the requested resource is denied. |
| 404 Not Found | The requested resource was not found. |
| **5xx Server Errors** | |
| 500 Internal Server Error | An error occurred on the server. |
| 502 Bad Gateway | Invalid response from the upstream server. |
| 503 Service Unavailable | The server is temporarily overloaded or under maintenance. |

## Error Handling

In case of errors (4xx, 5xx response), an appropriate error message will be returned, such as:

```
{
   "error": "Invalid phone number format"
}
```

```
{
   "error": "Invalid Transaction ID."
}
```

```
{
   "error": "Authentication failed, check your API key."
}
```

```
{
   "error": "A server-side issue occurred. Please retry your request."
}
```