REMITA LENDING SERVICE INTEGRATION DOCUMENT

Component Name	Remita Lending Service Integration
General Description	Outlines the API methods for integration to the Remita Lending Service.
Target Audience	Integration Partners

Table of Contents

SECTION 1 Overview

Remita is a multi-bank, modular e-Payments, e-Invoicing, e-Collections, e-Payroll, and e-Schedules delivery solution available on mobile (as an app downloadable to your phone/device (https://play.google.com/store/apps/details?id=com.systemspecs.remitamobile&hl=en) and web (www.remita.net) platforms. It remits and collects funds to/from accounts in any Financial Institution via various payment channels (e.g. Bank Branch, Credit/Debit Cards, Internet Banking, Mobile Wallets, etc.).

In implementing the Remita loan integration platform, SystemSpecs has provided a standard-based REST interface, which enables application developers interact in a powerful, yet secure way with Remita for the purpose of Banks and other loan providers making informed decisions as regards issuing loans to applicants. This document specifies the requirements for integrating to the Remita Lending Service.

The Remita loan integration API offers banks, loan issuers/providers a data intelligence platform that mitigates the risk factor involved in granting facility to loan applicants. Remita consists of a salary payment platform hosting a considerable amount of its customer's salary information. SystemSpecs shares this data with the loan issuer to enable them determine applicants' eligibility in terms of loan amounts they can realistically repay from their net salaries. The integration will also facilitate repayment of the loans by deducting the loan amounts from applicants' salary accounts when wages are paid.

A full implementation of the API methods in the following segments will enable the integrating partner (bank, loan issuer/provider) carry out an intelligent evaluation of the applicant's ability to repay requested amounts and manage repayment/recovery of the same.

The following sections discusses how you get started.

SECTION 2 Getting Started on Remita Lending Service

To get started on the Remita Lending Service (RLS), you are required to:

(a) **Register a profile on Remita**: You can visit www.remita.net to sign-up if you are not already registered as a merchant/biller on the platform.

(b) **Receive the Remita credentials that certify you as a Biller**: SystemSpecs will send you your merchant ID and an API Key necessary to secure your handshake to the Remita platform, upon completion of integrating the API to your portal.

All API parameters should be encrypted with AES 256 Algorithm using the shared keys. This value should be protected at all times and should not be shared. See Appendix A for more information on AES encryption. All API responses are formatted using JSend. (*JSend is a specification that specifies how JSON responses from web servers should be formatted. For more information see https://labs.omniti.com/labs/jsend*

A success status response from the API call means that all went well and (usually) some data was returned. You should check the content of the data returned for the service response typically denoted by "responseCode". You need to combine the base URLs below with appropriate endpoint URI, payloads and HTTP headers to make a call.

- **Test Environment Base URL**: http://www.remitademo.net/remita/exapp/api/v1/send/api/loansvc
- **Live Environment Base URL**: https://login.remita.net/remita/exapp/api/v1/send/api/loansvc

APIs available on the RLS include:

1. Request Salary Payment History
2. Loan Disbursement Notification
3. Stop Loan Collection
4. Mandate Payment History

The following segments discuss these in detail.

SECTION 3 Remita Lending Service APIs

The first step to granting loans via the RLS involves the issuer sending a request to Remita for the applicants' salary payment history. The execution of this method implies that the loan issuers' customers have opted into the service (consent has been obtained).

LA01: Request Salary Payment History

URI: /data/api/v2/payday/salary/history/ph

Method: POST

Request Headers

Header	Description
Content-Type	application/json
MERCHANT_ID	This is the merchant identifier
API_KEY	This is your token for authentication to Remita
REQUEST_ID	This uniquely identifies a request
AUTHORIZATION	Api authorization token (*see Appendix C for details of how to generate token)*

Fields and Values

Parameter Name	Description
authorisationCode	This is a unique code that implies the customer opt in for the service. This is to be provided by the loan provider. AuthorisationCodes are unique and cannot be requested or recycled. Error 13 will be returned in such cases.
authorisationChannel	How the authorization (Opt-In) was received from the customer. Options are USSD | WEB | MOBILE | BRANCH | PHONE | OTHER
phoneNumber	The phone number the customer was authorized with in the case of USSD authorization.

Sample Salary Payment History Request (Plaintext)

```
{
        "authorisationCode":"12343252324",
        "phoneNumber":"08126227772",
        "authorisationChannel":"USSD"
}
```

Sample Salary Payment History Response

```
{
  "status": “SUCCESS",
  “hasData”:true,
  "requestId": “123ADDC312342",
  "responseId": “123ADDC312342/523ADDC312342",
  "responseCode": “00",
  "requestDate": “2017-09-06T16:44:32.377Z",
  "responseDate": “2017-09-06T16:45:00.0Z",
  "responseMsg": “SUCCESS",
  "data": [{
      "accountNo": "1232029282",
      "bankCode": "057",
      "companyName": "Spring Nigeria",
      "countSalaryCr": “6”,
      "customerId": "SPR0571232029282",
      "customerName": “James Oni Peters",
      "salaryPaymentDetails": [
        {
          “amount”:"12000.00",
          "paymentDate": "2017-09-06T16:44:32.377Z",
        },
        {
          "amount": "12000.00",
          "paymentDate": "2017-08-06T16:44:32.377Z",
        },
        {
          "amount": "12000.00",
          "paymentDate": "2017-07-06T16:44:32.377Z",
        }
      ],
      “loanHistoryDetails”: [{
         "loanProvider":"Money box",
         "loanAmount":"10000",
         "outstandingAmount":"5000",
         “loanDisbursementDate":"10/12/2017",
         "repaymentFreq":"MONTHLY"
      }]
 }]
}
```

Possible Response Codes	
00	Successful
01	Failed

The loan issuer (provider/bank) completes an eligibility evaluation against the applicant using the salary history information retrieved from Remita and issues the facility accordingly.

LA02: Loan Disbursement Notification

The issuer/bank then informs SystemSpecs of loan amounts already granted to the applicant/customer. This notification automatically creates a Direct Debit mandate for the customer.

URI: /data/api/v2/payday/post/loan
METHOD: POST

Request Headers

Header	Description
Content-Type	**application/json**
MERCHANT_ID	**This is the merchant identifier**
API_KEY	**This is your token for authentication to Remita**
REQUEST_ID	**This uniquely identifies a request**
AUTHORIZATION	**Api authorization token (*see Appendix C for details of how to generate token)***

Fields and Values

Parameter Name	Description
customerId	A unique reference that identifies a salary earner within REMITA . This customerId would have been sent from Remita to the Bank's data enrichment service.
authorisationCode	This is a unique code that implies the customer opt in for the service. This is to be provided by the Bank. This is the same authorisation code that was sent to REMITA in LA01
authorisationChannel	How the authorisation (Opt-In) was received from the customer. Options are USSD | WEB | MOBILE | BRANCH | PHONE | OTHER
phoneNumber	The phone number tied to the account the loan was disbursed into.
accountNumber	The account number the loan was disbursed into.
currency	Currency in which loan was disbursed
loanAmount	Amount customer was given as loan
collectionAmount	Amount to be collected by REMITA the next time Salary is to be paid.
dateOfDisbursement	Date when the Loan was disbursed to the Customer
dateOfCollection	Tentative date which REMITA should start the process of collection
totalCollectionAmount	This is the total amount REMITA is collecting from the customer for loan disbursed.
numberOfRepayments	This is how many times the collectionAmount should the deducted for

Sample Loan Disbursement Notification Request (application/json)

```
{
  "customerId": "SPR0571232029282",
  "authorisationCode": "1234325232",
  "authorisationChannel":"USSD",
  "phoneNumber": “080512122211"
  "accountNumber": “0123123222”,
  "currency": "NGN",
  “loanAmount”:”10000.00",
  “collectionAmount”:”700.00",
  "dateOfDisbursement": "06-09-2018",
  "dateOfCollection": "06-09-2018",
  “totalCollectionAmount”:”10000.00",
  “numberOfRepayments”:”12"
}
```

Sample Loan Disbursement Respnse

```
{
        "status": “SUCCESS",
        "requestId": "223ADDC312342",
        "responseId": “223ADDC312342/623ADDC312342",
        "responseCode": “00",
        "requestDate": “2017-09-06T16:44:32.377Z",
        "responseDate": “2017-09-06T16:45:00.0Z",
        "responseMsg": “SUCCESS",
        "responseData": [{
                "authorisationCode": "1234325232",
                "customerId": "SPR0571232029282",
                "mandateReference": "REM0571232029282"
        }]
 }
```

Possible Response Codes	
00	Successful
01	Failed

SystemSpecs should now have been duly notified by the bank/provider as regards which applicants have been granted loans and the amounts disbursed to them.

SC01: Stop Loan Collection

This is used to request a STOP on the collection of a repayment amount due. This could happen if the customer has made payment for the said repayment via another means e.g. cheque or bank transfer or cash.

URI: /data/api/v2/payday/stop/loan

METHOD: POST

Request Headers

Header	**Description**
Content-Type	application/json
MERCHANT_ID	This is the merchant identifier
API_KEY	This is your token for authentication to Remita
REQUEST_ID	This uniquely identifies a request
API_DETAILS_HASH	Api authorization token (*see Appendix C for details of how to generate token)*

Fields and Values

Parameter Name	**Description**
customerId	A unique reference that identifies a salary earner within REMITA. This customerId would have be sent from Remita to the Banks data enrichment service.
authorisationCode	This is a unique code that implies the customer opt in for the service. This is to be provided by the Bank
mandateReference	A unique reference used to identify each loan repayment created.

Sample Stop Loan Collection Request (Plaintext)

```
{
        "authorisationCode": "1234325232",
        "customerId": "SPR0571232029282",
        "mandateReference": "REM0571232029282"
}
```

Sample Stop Loan Collection Response

```
{
	"status": “SUCCESS",
	"requestId": "223ADDC312342",
	"responseId": “223ADDC312342/623ADDC312342",
	"responseCode": “00",
	"requestDate": “2017-09-06T16:44:32.377Z",
	"responseDate": “2017-09 06T16:45:00.0Z",
	"responseMsg": “SUCCESS",
	"responseData": [{
	   "authorisationCode": "1234325232",
	   "customerId": "SPR0571232029282",
	   "mandateReference": "REM0571232029282"
	}]
}
```

Possible Response Codes	
00	Successful
01	Failed

MH01: Mandate Payment History

This is used to request a payment history on a mandate that has been set up via the loan disbursement endpoint.

URI: /data/api/v2/payday/loan/payment/history

METHOD: POST

Request Headers

Header	**Description**
Content-Type	‘application/json’
MERCHANT_ID	This is the merchant identifier
API_KEY	This is your token for authentication to Remita
REQUEST_ID	This uniquely identifies a request
AUTHORIZATION	'remitaConsumerKey=apiKey,remitaConsumerToken=apiHash”

Fields and Values

Parameter Name	Description
customerId	A unique reference that identifies a salary earner within REMITA. This customerId would have be sent from Remita to the Banks data enrichment service.
authorisationCode	This is a unique code that implies the customer opt in for the service. This is to be provided by the Bank
mandateReference	A unique reference used to identify each loan repayment created.

Sample Mandate Payment History Request (Plaintext)

```
{
  "authorisationCode":"1234325232",
  "customerId":"2001351392",
  "mandateRef":"230312222627"
}
```

Sample Mandate Payment History Response

```
{
	"status": "success",
	"hasData": true,
	"responseId": "1568972452448/1568972452448",
	"responseDate": "20-09-2019 09:40:59+0000",
	"requestDate": "20-09-2019 09:40:58+0000",
	"responseCode": "00",
	"responseMsg": "SUCCESS",
	"data": {
		"customerId": "00000758",
		"firstName": "Stanbic IBTC Pension Managers Ltd",
		"lastName": null,
		"phoneNumber": null,
		"loanMandateReference": "190213782484",
		"totalDisbursed": 20000.0,
		"outstandingLoanBal": 0.0,
		"loanRepaymentRef": null,
		"employerName": "GOAL GETTERS NIG. LTD",
		"salaryAccount": "0411428063180",
		"authorisationCode": "12qw121eeeeeeeqe111",
		"salaryBankCode": "ZENITH BANK PLC",
		"disbursementAccountBank": "ACCESS BANK PLC",
		"collectionStartDate": "25-09-2017 11:49:25+0000",
		"dateOfDisbursement": "25-09-2017 11:49:25+0000",
		"disbursementAccount": "04410010101",
		"status": "NEW",
		"lenderDetails": "SHELL",
		"repayment": [{
			"transactionamount": 21000.0,
			"deductiondate": "03-07-2019 16:56:36+0000",
			"paymentstatus": "TCLS"
		}]
	}
}
```

Possible Response Codes	
00	Successful
01	Failed

SECTION 4 Appendix

Appendix A

AES 256 Encryption

The test encryption parameters below have been used in encrypting data in the request body samples in this document. However, a different Vector and SecretKey pair may be provided upon commencement of integration.

Test Environment Encryption Parameters	
Vector/ ENC_INIT_VECTOR	044|044APIACCESS
SecretKey/ ENC_KEY	044APIACCESS|044
Algorithm	AES
Cipher	AES/CBC/PKCS5PADDING

See http://aesencryption.net/ for more information about the Advanced Encryption Standard (AES) encryption algorithm, and Appendix F, for sample codes on how to perform encryption for this integration.
If using java for encryption, it requires the Java Cryptography Extension (JCE) Unlimited Strength Policy Files. "Unlimited strength" policy files contain no restrictions on cryptographic strengths, in contrast to the "strong" but limited cryptography policy files bundled in a JRE.

Applying policy files:

- Download the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files from Oracle or IBM.
- Be sure to download the correct policy file updates for your version of Java (Java 7 or 8: http://www.oracle.com/technetwork/java/javase/downloads/index.html)
- Un-compress and extract the downloaded file. The download includes a Readme.txt and two .jar files with the same names as the existing policy files.
- Locate the two existing policy files: local_policy.jar US_export_policy.jar
- On UNIX, look in <java-home>/lib/security/
- On Windows, look in C:/Program Files/Java/jre<version>/lib/security/
- Replace the existing policy files with the unlimited strength policy files you extracted.

Appendix B

Transaction Response Codes

Codes	Description
01	Success
02	Transaction failed
03	Bad Request
04	Authentication Error
06	System Error
07	Limit Reached
08	Unknown Payment Reference
10	Insufficient Amount
12	Resource not found
13	Duplicate request
14	Authentication expired
15	User registration error
16	Duplicate user registration error
17	User verification required
22	Unknown transaction
23	Already processed transaction
24	Invalid request (missing headers or parameters)
25	Error processing request
26	Unknown merchant
27	Bank account not linked with merchant
28	Unknown bank account
29	Decryption error
30	Account pending authorization for merchant debit
31	Invalid credit account
32	Missing request parameters
33	Invalid credit account
34	Api authorization error (api hashing error)
35	Request timestamp expired
36	Invalid request timestamp format
37	Unauthorized credit account
38	Unauthorized debit account
39	Processing payments
40	Mandate setup failed
41	Activation failed
42	Unauthorized api access

Appendix C

HMAC-SHA256 Signatures for REST Requests

This section helps with understanding how to do the one-way hashing of all the http header values that are passed in your requests to our API.

The main http header parameters (they must be spelt as shown) are as follows: All the header parameters below are populated and sent for all requests except for the MERCHANT_ID parameter that is only sent for requests matching the url pattern - ***{BASE_URL}/rpg/api/v2/merc/**.***

- **HTTP HEADERS -** API_KEY MERCHANT_ID REQUEST_ID REQUEST_TS API_DETAILS_H ASH
- **HASHING KEY -** API_TOKEN (Not a header parameter. This is used as a one-way hashing key. It is concatenated with the API_KEY and REQUEST_ID http header parameters)

Authentication Parameters
For all requests, all these header parameters must be hashed using the one way hashing algorithm SHA-512. The values of the parameters must be concatenated as below. The source code for doing that is already provided in the accompanying sample Eclipse/STS Java Maven Project.

Given that the concatenated values below are variables that have been pre-assigned with values, their concatenation can be done as shown below when programming in Java.

API_DETAILS_HASH = HASHING_FUNCTION (API_KEY+REQUEST_ID+API_TOKEN)
Sample source code covering this hashing can be found in Appendix F.

Basic Authentication Processes:

The following describes the steps required to authenticate requests to AWS using an HMAC-SHA512 request signature.

1. You construct a request to the Remita Payment Gateway.
2. You calculate a keyed-hash message authentication code (HMAC-SHA512) signature with your secret access API token. For information about HMAC, see RFC2104.
3. You include the signature and the other request headers in the request, and then send the request to the Remita Payment Gateway.
4. The Remita Payment Gateway API uses your access API key (which is the same as our API's API_KEY http header) to look up your secret access API token.
5. Remita Payment Gateway API generates a signature from the request data and the secret access API.
   - Token with the same algorithm you used to calculate the signature you sent in the request.
6. If the signature generated by the Remita Payment Gateway matches the one you sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and the Remita Payment Gateway returns an error response.

Appendix D

Request Timestamp

The time stamp (or expiration time) you use in the request must be a dateTime object, with the complete date including hours, minutes, and seconds (for more information, see https://www.w3.org/TR/xmlschema-2/#dateTime). For example: 2007-01-31T23:59:59Z. Although it's not required, we recommend you provide the time stamp in the Coordinated Universal Time (Greenwich Mean Time) time zone.

The request automatically expires 5 minutes after the time stamp (in other words, Remita does not process a request if the request time stamp is more than 5 minutes earlier than the current time on Remita servers). Make sure your server's time is set correctly.

Important

If you're using .NET you must not send overly specific time stamps, due to different interpretations of how extra time precision should be dropped. To avoid overly specific time stamps, manually construct dateTime objects with no more than millisecond precision.

Appendix E

Supported Currency Codes

Currency	CurrencyCode
Nigerian Naira	NGN

Appendix F

Sample Encryption and HMAC/One-Way Hashing Source Code

The Sample code for the HMAC/one-way hashing, and the two-way encryption and decryption, are available at the URL below.

https://www.remita.net/developers/assets/sample-codes/EncryptionsCodesSampleMvn.zip