# LANNISTER PAY (NodeJS Assessment)

Payment processing involves several components / services. One of such is determining the processing fee to charge per transaction. This assessment is about implementing an NGN (Nigerian Naira) fee processing service for a fictional Payment Processor (LannisterPay).

LannisterPay has reached out to you to help implement a transaction fee processing service. This service is meant to calculate the fee applicable to a transaction based on specific fee configurations they will share with you.

LannisterPay uses a custom fee configuration spec (FCS) to describe applicable fees.

## Fee Configuration Spec (FCS)

The LannisterPay custom FCS defines (line by line) fee entries that can be applied to a given transaction. An example is shared below:

LNPY1221 NGN LOCL CREDIT-CARD(*) : APPLY PERC 1.4
LNPY1222 NGN INTL CREDIT-CARD(MASTERCARD) : APPLY PERC 3.8
LNPY1223 NGN INTL CREDIT-CARD(*) : APPLY PERC 5.8
LNPY1224 NGN LOCL USSD(MTN) : APPLY FLAT_PERC 20:0.5
LNPY1225 NGN LOCL USSD(*) : APPLY FLAT_PERC 20:0.5

## Basic FCS (Single Line) Syntax

{FEE-ID} {FEE-CURRENCY} {FEE-LOCALE} {FEE-ENTITY}({ENTITY-PROPERTY}) : APPLY {FEE-TYPE} {FEE-VALUE}

# FCS Syntax keywords

- {FEE-ID} Each fee configuration has an 8-Character (Alphanumeric) Identifier.
- {FEE-CURRENCY} - Currency the fee configuration is applicable to. (Will be NGN for this assessment)
- {FEE-LOCALE} Locale the fee configuration is applicable in. Locales are determined by two main attributes of a transaction, the currency and the country of the entity used for payment. The rule is, if the currency's country and the country of the payment entity are the same, the transaction is processed as a local one (**LOCL**) e.g. an NGN 5000 transaction with an NG-Issued GTBANK Mastercard is local. If the currency's country and the country of the payment entity are not the same, the transaction is processed as an international one (**INTL**) e.g. an NGN 5000 transaction with a US-Issued AMEX card is international.
- {FEE-ENTITY} The entity to be charged for the transaction. This could be a credit / debit card, a USSD mobile number, a Nigerian Bank account OR a Wallet ID. The supported values are: CREDIT-CARD, DEBIT-CARD, BANK-ACCOUNT, USSD, WALLET-ID
- {ENTITY-PROPERTY} This is used, primarily, to define specificity. It refers to any of the valid payment entity properties. A value of * means it applies to all. Examples:
    - LNPY1222 NGN INTL CREDIT-CARD(MASTERCARD) : APPLY PERC 3.8 means this fee specification is applicable only to MASTERCARD CREDIT CARDS
    - LNPY1221 NGN LOCL CREDIT-CARD(*) : APPLY PERC 1.4 means this fee specification is applicable to all CREDIT CARDS
    - LNPY1224 NGN LOCL USSD(MTN) : APPLY FLAT_PERC 20:0.5 means this fee specification is only applicable to USSD transactions where the mobile number used is from the MTN telco provider.
- {FEE-TYPE} The type of the fee defines how it is applied. Possible values for the FEE-TYPE are:
    - FLAT A fee of type flat is applied as is. E.g. FLAT 50 means 50 NGN is applied directly as the transaction fee.
    - PERC A fee of type percentage is applied differently. The value is multiplied by the transaction amount and divided by 100 to get the applicable fee value. E.g. PERC 1.4 means 1.4% of the transaction amount is applied as the transaction fee.
    - FLAT_PERC A fee of type flat and percentage indicates that the fee entry has both FLAT and percentage values. E.g. FLAT_PERC 20:0.5 means a flat fee of 20 NGN and 0.5% of the transaction amount is applied as the transaction fee.
- {FEE-VALUE} The value of the fee to be applied. It's usually a non-negative numeric (0 inclusive) value with the exception of cases where {FEE-TYPE} is FLAT_PERC, then it's a string in the form {FLAT-VALUE}:{PERC-VALUE} e.g. FLAT_PERC 20:0.5. Examples:
    - FLAT 50 means 50 NGN is applied directly as the transaction fee

- PERC 1.4 means 1.4% of the transaction amount is applied as the transaction fee.
- FLAT_PERC 20:0.5 means a flat fee of 20 NGN and 0.5% of the transaction amount is applied as the transaction fee

## Some additional information from LannisterPay

1. The {FEE-CURRENCY} {FEE-LOCALE} {FEE-ENTITY} can also have their values set as *. An example, LNPY1224 * * CREDIT-CARD(*) : APPLY FLAT_PERC 20:0.5 means the fee should be applied to all CREDIT-CARD transactions in any currency and locale.
2. Only one fee configuration entry is applicable to a transaction at any time. Your service is to decide the most apt configuration to apply based on its precedence.
3. The higher the specificity of a fee configuration entry, the higher its precedence. E.g. LNPY1229 NGN INTL CREDIT-CARD(MASTERCARD) : APPLY PERC 3.8 has a higher precedence than LNPY1221 NGN INTL CREDIT-CARD(*) : APPLY PERC 5.8. If a MASTERCARD CREDIT-CARD transaction is presented to your service, the first configuration should be applied even though both of them are valid CREDIT CARD configurations. This is because it's more specific (in "specifying" that it's applicable to only "Mastercard" transactions). In the same vein, LNPY1221 NGN INTL CREDIT-CARD(*) : APPLY PERC 5.8 has a higher precedence than LNPY1222 NGN INTL *(*) : APPLY PERC 5.8 because it's more specific (in "specifying" that it's applicable to credit card transactions). Configuration entries with more specific details (be it the ENTITY TYPE / PROPERTY OR LOCALE) should be considered first while configurations having a lot of * values should be considered last.

## Your task

LannisterPay wants you to create a NodeJS API service. The API service should have the endpoints described below.

### Fee setup endpoint

HTTP POST /fees This endpoint should accept data containing a valid fee configuration spec. It should parse the specification and store the result in a way that makes it easy for your application to reference later. Some options to explore include storing in a json file

on your API server, using Redis / MongoDB / any other storage system of your choosing.

The data structure you decide to parse the fee configuration spec into is entirely up to you. You can decide to use an array, a key-value pair dictionary, hashmap e.t.c.

**Sample Post Data**


```
{
   "FeeConfigurationSpec": "LNPY1221 NGN * *(*) : APPLY PERC 1.4\nLNPY1222 NGN I
NTL CREDIT-CARD(VISA) : APPLY PERC 5.0\nLNPY1223 NGN LOCL CREDIT-CARD(*) :
APPLY FLAT_PERC 50:1.4\nLNPY1224 NGN * BANK-ACCOUNT(*) : APPLY FLAT 100\nL
NPY1225 NGN * USSD(MTN) : APPLY PERC 0.55"
}
```


**Sample Response**


```
HTTP 200 OK
{
  "status": "ok"
}
```


**Fee computation endpoint**

HTTP POST /compute-transaction-fee This endpoint should accept a single transaction payload and use the data you stored from the HTTP POST /fees endpoint described above to compute the fee applicable to the transaction. The transaction payload is an object containing the following fields:

- ID The unique id of the transaction.
- Amount The non-negative, numeric transaction amount.
- Currency The transaction currency.
- CurrencyCountry Country the transaction currency is applicable in. Useful for determining the transaction locale.
- Customer An object containing the customer information. It has the following fields:
    - ID Unique id of the customer .
    - EmailAddress Email address of the customer.
    - FullName Full name of the customer.
    - BearsFee Indicates whether or not the customer is set to bear the transaction cost. If this is true, the final amount to charge the customer is Amount + ApplicableFee, if not, the customer is charged the same value as the transaction amount.
- PaymentEntity An object representing the payment entity to be charged for the transaction. It has the following fields:
    - ID - Unique id of the entity.

- Issuer - The issuing company / organization for the entity e.g. Banks, Telco Providers / Wallet Service Providers.
- Brand - Applicable only to card-type transactions e.g. MASTERCARD, VISA, AMEX, VERVE e.t.c.
- Number The payment entity number (masked pan in case of credit/debit cards, bank account numbers, mobile numbers, wallet ids e.t.c.)
- SixID The first six digits of the payment entity number.
- Type The type of the entity e.g. CREDIT-CARD, DEBIT-CARD, BANK-ACCOUNT, USSD, WALLET-ID
- Country The issuing country of the entity e.g. NG, US, GH, KE e.t.c. It's used together with the CurrencyCountry to determine a transaction's locale.

The values for ID, Issuer, Brand, Number and SixID can be passed as {ENTITY-PROPERTY} values in fee configurations to add more specificity. E.g.

LNPY8222 NGN INTL CREDIT-CARD(MASTERCARD) : APPLY PERC 3.8
LNPY8223 NGN LOCL CREDIT-CARD(GTBANK) : APPLY PERC 2.8
LNPY8224 NGN LOCL CREDIT-CARD(530191******2903) : APPLY PERC 3.8
LNPY8225 NGN LOCL USSD(MTN) : APPLY PERC 3.8
LNPY8226 NGN LOCL USSD(08032211002) : APPLY PERC 3.8

**Sample payload:**


```
{
  "ID": 91203,
  "Amount": 5000,
  "Currency": "NGN",
  "CurrencyCountry": "NG",
  "Customer": {
    "ID": 2211232,
    "EmailAddress": "anonimized29900@anon.io",
    "FullName": "Abel Eden",
    "BearsFee": true
  },
  "PaymentEntity": {
    "ID": 2203454,
    "Issuer": "GTBANK",
    "Brand": "MASTERCARD",
    "Number": "530191******2903",
    "SixID": 530191,
    "Type": "CREDIT-CARD",
    "Country": "NG"
  }
}
```


The endpoint should respond with an object having the following fields:

- AppliedFeeID ID of the fee applied to the transaction.
- AppliedFeeValue Computed value of the applied fee. Note that, for flat fees, this is the same as the fee value. For percentage fees, this is obtained by doing ((fee value

* transaction amount ) / 100) and for FLAT_PERC types, this is obtained by doing flat fee value + ((fee value * transaction amount ) / 100). An example, if the transaction amount is 1500:
    - For configuration LNPY0221 NGN LOCL CREDIT-CARD(*) : APPLY PERC 1.4, AppliedFeeValue = (1.4 / 100) * 1500
    - For configuration LNPY0222 NGN LOCL CREDIT-CARD(*) : APPLY FLAT 140, AppliedFeeValue = 140
    - For configuration LNPY0223 NGN LOCL CREDIT-CARD(*) : APPLY FLAT_PERC 140:1.4, AppliedFeeValue = 140 + ((1.4 / 100) * 1500)
- ChargeAmount The final amount to charge the customer for the transaction. The value is dependent on what Customer.BearsFee is set to.
    - If Customer.BearsFee is true, ChargeAmount = Transaction Amount + AppliedFeeValue
    - If Customer.BearsFee is false, ChargeAmount = Transaction Amount
- SettlementAmount The amount LannisterPay will settle the merchant the transaction belongs to after the applied fee has been deducted. In essence: SettlementAmount = ChargeAmount - AppliedFeeValue

**Sample response below:**

```
HTTP 200 OK
{
  "AppliedFeeID": "LNPY0222",
  "AppliedFeeValue": 230,
  "ChargeAmount": 5230,
  "SettlementAmount": 5000
}
```

If none of the fee configurations is applicable to a transaction, you should return with a properly formatted error message and a non-200 HTTP status code.

LannisterPay has also requested that the ideal response time for the /compute-transaction-fee endpoint is 50ms (Milliseconds) or less. How you choose to store the fee configuration after parsing the specification goes a long way in determining how fast your endpoint will respond. *You are also allowed to use any other methods / interesting tricks to speed up your response time.*

## Task Submission

Once done with your implementation, you can submit a link to your API using this google forms link. (It will be open till the 21st of March, 2022)

# FAQS

**#1**

**I want to use Heroku to host my API, won't this affect my response time seeing as the free version sleeps after a cerain idle time.**

*This will not be an issue, the "/fees" endpoint will be called first before any other tests are run on your API. Since the tests on your API will be run in one session, the call to "/fees" should be sufficient to warm up your Heroku dyno[s]*

**#2**

**Can I use typescript?**

*Yes, you can - Vanilla Javascript is preferred though.*

**#3**

**Am I allowed to use 3rd party frameworks and or libraries?**

*Yes, you can use any framework or library you want.*

**#4**

**How extensive can I make the specification parser?**

*This is entirely up to you. The choice of data structure, be it an array, hashmap, list e.t.c. whether or not you choose to store in a flat file, a redis instance, NoSQL database e.t.c. All these are for you to decide. The key thing to bear in mind when deciding how to implement your parser or transform the fee configuration spec is this, the "/compute-transaction-fee" is expected to respond in 50ms or less.*

**#5**

**Should the API use any form of authentication or authorization like tokens or secret keys?**

*No, this isn't a requirement.*

# Test Examples (Data)

LannisterPay shared example data (request payloads and expected responses) that you can use to test your implementation

Please note that these are just for you to test. After submission, your API will be tested with a different set of request payloads.

**Example FCS**

LNPY1221 NGN * *(*) : APPLY PERC 1.4
LNPY1222 NGN INTL CREDIT-CARD(VISA) : APPLY PERC 5.0
LNPY1223 NGN LOCL CREDIT-CARD(*) : APPLY FLAT_PERC 50:1.4
LNPY1224 NGN * BANK-ACCOUNT(*) : APPLY FLAT 100
LNPY1225 NGN * USSD(MTN) : APPLY PERC 0.55

**Example FCS setup request and expected response**

Request
Expected Response


```
{
  "FeeConfigurationSpec": "LNPY1221 NGN * *(*) : APPLY PERC 1.4\nLNPY1222 NGN INTL CREDIT-CARD(VISA) : APPLY PERC 5.0\nLNPY1223 NGN LOCL CREDIT-CARD(*) : APPLY FLAT_PERC 50:1.4\nLNPY1224 NGN * BANK-ACCOUNT(*) : APPLY FLAT 100\nLNPY1225 NGN * USSD(MTN) : APPLY PERC 0.55"
}
```


**Example Fee Computation I**

Request
Expected Response
Response Explanation


```
{
   "ID": 91203,
   "Amount": 5000,
   "Currency": "NGN",
   "CurrencyCountry": "NG",
   "Customer": {
      "ID": 2211232,
      "EmailAddress": "anonimized29900@anon.io",
      "FullName": "Abel Eden",
      "BearsFee": true
   },
   "PaymentEntity": {
      "ID": 2203454,
      "Issuer": "GTBANK",
      "Brand": "MASTERCARD",
      "Number": "530191******2903",
```

```
      "SixID": 530191,
      "Type": "CREDIT-CARD",
      "Country": "NG"
   }
}
```


### Example Fee Computation II

Request
Expected Response
Response Explanation


```
{
   "ID": 91204,
   "Amount": 3500,
   "Currency": "NGN",
   "CurrencyCountry": "NG",
   "Customer": {
      "ID": 4211232,
      "EmailAddress": "anonimized292200@anon.io",
      "FullName": "Wenthorth Scoffield",
      "BearsFee": false
   },
   "PaymentEntity": {
      "ID": 2203454,
      "Issuer": "AIRTEL",
      "Brand": "",
      "Number": "080234******2903",
      "SixID": 080234,
      "Type": "USSD",
      "Country": "NG"
   }
}
```


### Example Fee Computation III

Request
Expected Response
Response Explanation


```
{
   "ID": 91204,
   "Amount": 3500,
   "Currency": "USD",
   "CurrencyCountry": "US",
   "Customer": {
      "ID": 4211232,
      "EmailAddress": "anonimized292200@anon.io",
      "FullName": "Wenthorth Scoffield",
      "BearsFee": false
   },
   "PaymentEntity": {
```

```
    "ID": 2203454,
    "Issuer": "WINTERFELLWALLETS",
    "Brand": "",
    "Number": "AX0923******0293",
    "SixID": "AX0923",
    "Type": "WALLET-ID",
    "Country": "NG"
  }
}
```