|  |  |
| --- | --- |
| **Project Name:** | BnE Pending Authorization |
| **Project No.:** |  |
| **Document Owner**  **(Name / SOE ID):** | Francisco Serrato  FS44512 |
| **System Name(s):** | Wholesale eBanking |

GCT Technical Design (TD) Micro-service

Wholesale eBanking BnE Pending Authorization

Transactions

Tabla de Contendo

[Document Change Control 3](#_Toc522121476)

[Review and Approval Table 3](#_Toc522121477)

[Overview 4](#_Toc522121478)

[Micro Service Description 4](#_Toc522121479)

[Virtualized Service 5](#_Toc522121480)

[Repo Details 5](#_Toc522121481)

[Functionality Description 5](#_Toc522121482)

[Assumptions, Constraints & Decisions 6](#_Toc522121483)

[Assumptions: 6](#_Toc522121484)

[Constraints: 6](#_Toc522121485)

[Decisions 6](#_Toc522121486)

[System Logic Flowchart 7](#_Toc522121487)

[Operating Environment Description 8](#_Toc522121488)

[Design Alternatives 8](#_Toc522121489)

[Reusable Asset 8](#_Toc522121490)

[Software Tools for Development (Required) 9](#_Toc522121491)

[API Specification 9](#_Toc522121492)

[1. pendingAuthorization 9](#_Toc522121493)

[2. retrieveTransactionDetails 12](#_Toc522121494)

[Exception Flow EF1 17](#_Toc522121495)

[Headers 17](#_Toc522121496)

[System Interfaces (Required) 20](#_Toc522121497)

[Legacy Contracts Definitions 20](#_Toc522121498)

[Input Specifications 20](#_Toc522121499)

[Output Specifications 20](#_Toc522121500)

[Security Specifications 20](#_Toc522121501)

[Additional Design Activities 20](#_Toc522121502)

[Glossary 20](#_Toc522121503)

[Appendix 21](#_Toc522121504)

## Document Change Control

| **Version No.** | **Revised By** | **Revision Date** | **Sections** | **Description of Change(s)** |
| --- | --- | --- | --- | --- |
| 0.1 | Francisco Serrato | 17/07/2018 | All | Initial |
| 0.2 | Francisco Serrato | 02/08/2018 | Add API | retreieveTransactionDetails |
| 0.3 | Francisco Serrato | 09/08/2018 | Modify | Additional Information |
| 0.4 | Francisco Serrato | 14/08/2018 | Add Sql | Include query for extract info of the db |
|  |  |  |  |  |

## 

## Review and Approval Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CREATION (SA)** | | | | |
| **Prepared By:** | **Name:** | **Francisco Serrato** | **Date:** | **13/07/2018** |
| **REVIEW (CL)** | | | | |
| **Reviewed By:** | **Name:** | **Carlos Velez** | **Date:** |  |
| **APPROVAL** | | | | |
| **Solutioning (SL)** | | | | |
| **Approved By:** | **Name:** | **Ricardo Cruz** | **Date:** |  |
| **Tech Lead (TL)** | | | | |
| **Approved By:** | **Name:** | **Ivan Hernandez / Ivan Fajardo** | **Date:** |  |
|  | | | | |

## Overview

As part of the Citibanamex service improvement strategy, it is required to obtain the information for the business banking from the pending registrations to be authorized, which are classified in: by file and individual, for this it is necessary to build a service which can provide this information online. This transformation is aligned with the use of micro-services technology, providing horizontal scalability and a more robust service.

## Micro Service Description

This micro service has 1 end point, which has the responsibility of recovering the pending records to be authorized either by file or by individual, for this it is necessary to recover this information from the eBanking database; these functionalities are grouped in the same micro-service.

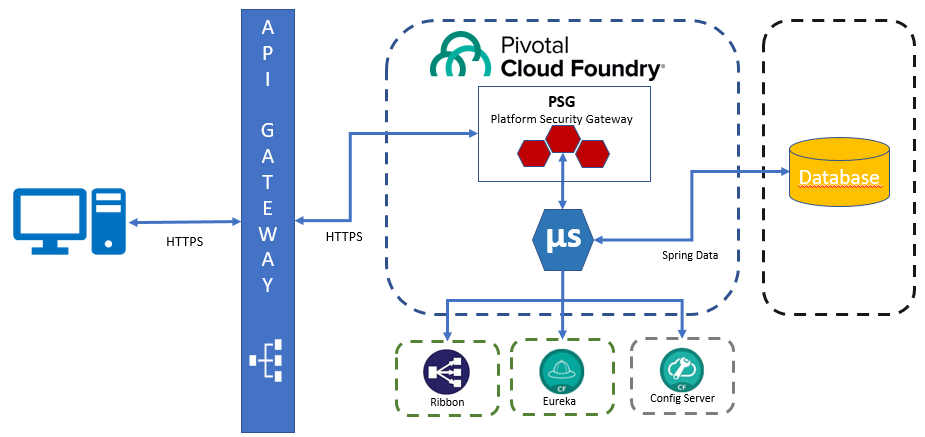


Fig 1. Micro service representation

## Virtualized Service

|  |  |
| --- | --- |
| **Name** | **Link** |
| **CHNN-BNE-V-Transactions** | <http://sd-fa73-6ad0:10038>   1. /api/v1/channels/bne/accounts/transactions/retrieve 2. /api /v1/channels/bne/transactions/retrieve |



## Repo Details

|  |  |
| --- | --- |
| **Name** | **Link** |
| Jira | <https://gct-cedt-pilot.nam.nsroot.net:8080/jira/browse/MAFF-704> |
| Confluence | <https://gct-cedt-pilot.nam.nsroot.net:8090/confluence/display/MRAF/BNE+-+Platform+Administration?src=contextnavpagetreemode> |
| Bitbucket  Swagger | <https://cedt-gct-bitbucket.nam.nsroot.net/bitbucket/projects/MAFIM/repos/mafim-v-api-swaggers/browse/mx-channels/CHNN-BNE-V-Transactions.json?at=refs%2Fheads%2Fdevelopment> |
| Bitbucket  Code |  |

## Functionality Description

After accessing the system through the login, you can check the pending transactions to be authorized, for this it is necessary to execute an API that is in charge of recovering this information. This micro-service accesses the database as an inherited system.

**APIs Description (Required)**

|  |  |
| --- | --- |
| **Name** | **Description** |
| **pendingAuthorization**  **Intake**  **[GetPednding AutorizationTx]** | Retrieves the total of transactions and operations pending of authorization. |
| **pendingAuthorizationDetails**  **Intake**  **[GetPednding Autorization]** | to retrieve transaction details from the SQL database. |
|  |  |

## Assumptions, Constraints & Decisions

For these APIs we have the next:

## Assumptions:

* It should be reusable services and it should return Pending authorization Counters
* The information will be retrieved of the eBanking database.

## Constraints:

* Platform Security Gateway (**PSG**) will validate the access security for each domain API, if the sid attribute is valid, the process continues with the execution of the API, otherwise, **PSG** will generate a response with the corresponding error.
* Resilience. Is mandatory, to manage the micro service resilience, should be implement **Hystrix** circuit breaker pattern.
* All development will be aligning throw API factory guidelines on any specific point not covered should be presented and approved by API factory committee.

## Decisions

* Use Spring Boot & Config Server for all micro-service configuration.
* Must use Spring Data JPA.
* Use connection pool for database access.
* Implement Eureka & Histryx.

## System Logic Flowchart

The next diagrams show functionality and stablished architecture for the APIs development.

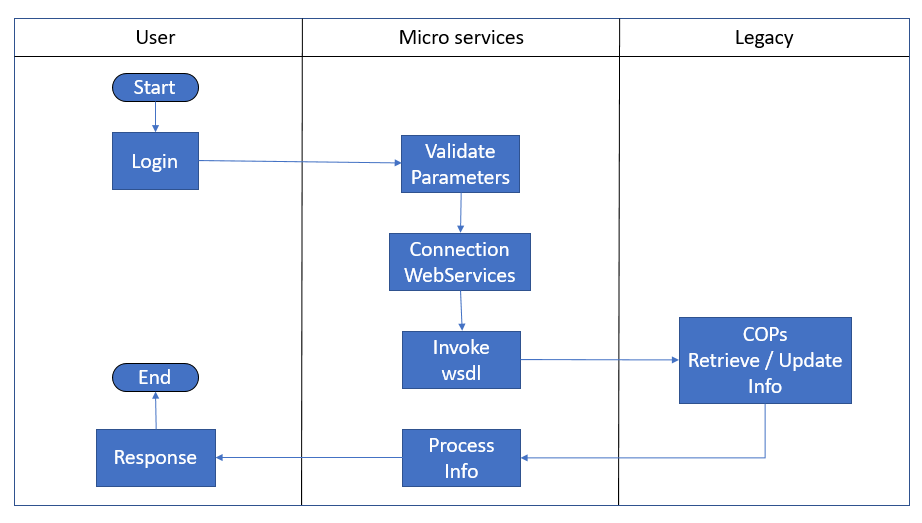


Fig 2. Flowchart diagram

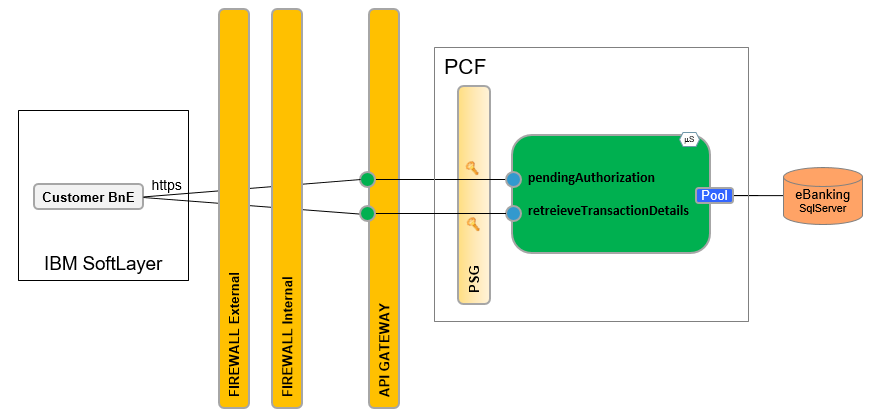


Fig 3. Component diagram

## Operating Environment Description

The environment is a Spring Boot micro-service that runs on PCF with a Spring Cloud data flow for the connection to the eBanking.

The following describes the deployment environments as well as the configuration attributes required for each one:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Component/Environment** | **DEV** | **SIT** | **UAT** | **PERF** | **PROD** |
| PCF | X | X | X | X | X |
| eBanking | X | X | X | X | X |
|  |  |  |  |  |  |

For detailed environment description, please refer to API Factory Environment’s guides.

## Design Alternatives

As functionality and NFR Requirements, the design perspective will be cover as dialogue invocation.

|  |  |
| --- | --- |
| NFR | Detail |
| TpS | Max 3 |
| Response Time | < 2 seg approx. |



## Reusable Asset

This micro-service is thinking of being reused by the different channels that require information from the business banking.

## Software Tools for Development (Required)

|  |  |
| --- | --- |
| Component Name | CTC ID / Location |
| Spring Tool Suite 3.7.2 | 87313 |
| SoapUI 5.3.0 | 94811 |
| Java SE Development Kit 8u141 | 96816/96814 |
|  |  |

## API Specification

It contains 1 API in charge of recovering the information of the pending transactions to authorize, classified in: by file and individual, the API only recovers 2 meters. For more details, it is described with a description and a diagram:

### pendingAuthorization

The objective of this API is to recover the number of records or the counter of pending transactions to be authorized, taking into account that we have 2 classifications, 1) File and 2) Individual; for this, it is necessary to connect to the eBanking Sql Server database using Spring Data JPA, using a pool of connections to make the connection more efficient.

**Detailed API sequence diagram**



Fig 4. API Sequence diagram

**Process Flow**

|  |  |
| --- | --- |
| **Step** | **Action** |
| 01 | * This API must validate session, If is valid continue…   If the session is not valid, execute the exception flow EF1 with the HTTP status code 401 and continue to step 07 |
| 02 | * This API receive 2 parameters:   {customerId, operatorId}  Retrieve the pending transaction number by authorization |
| 03 | * If the request is valid, continue… * If the request is not valid, execute the exception flow EF1 with the HTTP status code 400 and continue to step 07 |
| 04 | * Get a connection to the database eBanking * If there is a problem with connection, execute the exception flow EF1 with the HTTP status code 500 and continue to step 07 |
| 05 | * Execute the query to retrieve the pending transaction number by authorization * If there is problem with the info, execute the exception flow EF1 with the HTTP status code 400 and continue to step 07 |
| 06 | * Transform the database resultSet into a Json object. |
| 07 | * Create response object with the error/ result |
| 08 | * Return the response to User API |
| 09 | * End of flow. |



Fig 5. Activity diagram

**Service Interface**

Name: ***pendingAuthorization***

Method: **POST**

Path: /v1/channels/bne/accounts/transactions/retrieve

**Parameters**

|  |  |
| --- | --- |
| customerId **\*** | Unique identifier of the customer |
| operadorId **\*** | (before: Identifier of legal representative) Is the person who operates the BnE system, assigned by the company. |
| **\*** Parameters required | |

**Responses**

|  |  |
| --- | --- |
| **Code** | **Description** |
| 200 | Successful, when all in the API finished ok.  Example Value | Model  {  "totalPendingTransactions": 10,  "totalPendingOperations": 12  } |

**Additional Information**

N/A

### retrieveTransactionDetails

The objective of this API is to retrieve the details of pending records to be authorized, considering that we have 2 classifications 1) File and 2) Individual; for this, it is necessary to connect to the **eBanking** Sql Server database using Spring Data JPA, using a pool of connections to access the data. All the requred information exists in the same table and to carry out the process of identification of records it will be necessary to validate the *group\_value* field and compare it with the values that I send business; implementing the rule that everything that matches is **File** and the rest is **Individual**.

**Detailed API sequence diagram**



Fig 4. API Sequence diagram

**Process Flow**

|  |  |
| --- | --- |
| **Step** | **Action** |
| 01 | * This API must validate session, If is valid continue…   If the session is not valid, execute the exception flow EF1 with the HTTP status code 401 and continue to step 08 |
| 02 | * This API receive 3 parameters:   {customerId, operatorId, transactionClassificationCode}  Retrieve the pending transaction number by authorization |
| 03 | * If the request is valid, continue… * If the request is not valid, execute the exception flow EF1 with the HTTP status code 400 and continue to step 08 |
| 04 | * Get a connection to the database eBanking * If there is a problem with connection, execute the exception flow EF1 with the HTTP status code 500 and continue to step 08 |
| 05 | * Execute the query to retrieve the details pending authorization transaction * If there is problem with the info, execute the exception flow EF1 with the HTTP status code 400 and continue to step 08 |
| 06 | * Identify the type of record File/Individual, use the field ***value\_group*** with the values **06** and **20** for File, any other will be Individual |
| 07 | * Transform the database resultSet into a Json object. |
| 08 | * Create response object with the error/ result |
| 09 | * Return the response to User API |
| 10 | * End of flow. |



Fig 5. Activity diagram

**Service Interface**

Name: ***retrieveTransactionDetails***

Method: **POST**

Path: /v1/channels/bne/accounts/transactions/retrieve

**Parameters**

|  |  |
| --- | --- |
| customerId **\*** | Unique identifier of the customer |
| operadorId **\*** | (before: Identifier of legal representative) Is the person who operates the BnE system, assigned by the company. |
| transactionClassificationCode **\*** | Classification of transaction whether 1 =countable or 2 = non-countable (Type) |
| **\*** Parameter Required | |

**Responses**

|  |  |
| --- | --- |
| **Code** | **Description** |
| 200 | Successful, when all in the API finished ok.  Example Value | Model  {  "transactions": [  {  "transactionShortDescription": "PAGTERPARCHTAHU\_BE",  "transactionDescription": "PAGOS A TERCEROS CHEQUES - TARJETAS",  "transactionStatus": "Rechazada",  "transactionStatusCode": "012",  "transactionAmount": 1234.09,  "currencyCode": "MXN",  "authorizationCode": "011003",  "transactionDate": "20181226",  "transactionTime": "11:22:33",  "transactionResponseReasonCode": "Fecha Vencida",  "retrievedBy": "File",  "operatorId": "12",  "tableSerialNumber": "1",  "lastUpdatedBy": "Mary Sinkins",  "representatives": [  {  "operatorName": "Mary Kom"  }  ],  "destinationAccount": {  "branchId": "123456",  "accountNumber": "1234567890",  "accountHolderName": "Bryan Sanger"  },  "sourceAccount": {  "branchId": "1234578",  "accountNumber": "123567890",  "accountHolderName": "Bryan Singer"  },  "TransactionReferenceDetail": [  {  "transactionReference": "Nomina Banamex"  }  ]  }  ]  } |

**Additional Information**

* Attach business email to clarify the process to identify **File**/**Individual**; use the **group\_value** field of the table, where values **06** and **20** identify the File scenario.



* Attach example and structure table



* The query that can be used to retrieve the info would be:



* Is necessary apply this rules for arrays in the Swagger:
  + Query fields rep1, rep2 vs Swagger
    - Array index 0 -> rep1
    - Array index 1 -> rep2
  + Query fields bt\_char1, bt\_char2, bt\_char3, bt\_refnum1, bt\_refnum2 vs Swagger
    - Array index 0 -> bt\_char1
    - Array index 1 -> bt\_char2
    - Array index 2 -> bt\_char3
    - Array index 3 -> bt\_refnum1
    - Array index 4 -> bt\_refnum2

## Exception Flow EF1

| Step | User API Actions |
| --- | --- |
| 1 | Catch any exception. |
| 2 | Create an **ErrorResponse** object with the HTTP status code information corresponding, for example:  {  "type": "error",  "code": "string",  "details": "string",  "location": "string",  "moreInfo": "string",  "uuid": "string",  "timestamp": "string"  } |

## Headers

|  |  |
| --- | --- |
| **client\_id** \*  string (header) | Client ID generated during application registration |
| **Authorization** \*  string (header) | The Authorization Token received during login |
| **Accept** \*  string (header) | Content-Types that are acceptable for the response |
| **uuid** \*  string (header) | 128 bit UUID that you generate for every request |
| **Accept-Language**  string (header) | List of acceptable human languages for response |
| **Content-Type** \*  string (header) | Content-Types that are sent in the request |
| **channelId** \*  string (header) | Channel where request originated |
| **sid** \*  string (header) | SessionId sent by Consumer |

**Error Handling**

|  |  |
| --- | --- |
| **Code** | **Description** |
| 400 | Bad request, when the API has tried to process the request, but some aspect is not correct.  Example Value | Model  {  "type": "error",  "code": "string",  "details": "string",  "location": "string",  "moreInfo": "string",  "uuid": "string",  "timestamp": "string"  } |
| 401 | Unauthoritation, when the API cannot continue due to security issues or authorizations.  Example Value | Model  {  "type": "error",  "code": "string",  "details": "string",  "location": "string",  "moreInfo": "string",  "uuid": "string",  "timestamp": "string"  } |
| 403 | Forbidden, when the API tries to access a resource to which it does not have access.  Example Value | Model  {  "type": "error",  "code": "string",  "details": "string",  "location": "string",  "moreInfo": "string",  "uuid": "string",  "timestamp": "string"  } |
| 404 | Not Found, when the URL for the API does not exist or the resource to retrieve is empty.  Example Value | Model  {  "type": "error",  "code": "string",  "details": "string",  "location": "string",  "moreInfo": "string",  "uuid": "string",  "timestamp": "string"  } |
| 500 | Internal Server Error, when this happens it could indicate some problem with the request or a problem in the code of the server side.  Example Value | Model  {  "type": "error",  "code": "string",  "details": "string",  "location": "string",  "moreInfo": "string",  "uuid": "string",  "timestamp": "string"  } |

## System Interfaces (Required)

The following URIs will be required to be created on each environment (Dev, Sit, UAT, PERF):

## Legacy Contracts Definitions

N/A

## Input Specifications

See expected input in swagger contract definition.

## 

## Output Specifications

See expected output in swagger contract definition.

## Security Specifications

Follow the API Gateway security standard.

## Additional Design Activities

The standards and tools that should be use are described in the architecture document.



## Glossary

|  |  |
| --- | --- |
| PCF: | Pivotal Cloud Foundry - PaaS Platform. |
| PSG: | Platform Security Gateway. Micro service responsible to manage all API security. |
| Config Server | Every micro-service must implement a centralized configuration in a Git repository for proper administration. Spring Boot and Spring Cloud. |
| Discovery Service | Service discovery is the automatic detection of devices and services offered by these devices on a computer network. Service discovery aims to reduce the configuration efforts from users. |
| Circuit Breaker | It is a library designed to isolate access points to remote systems, services and third-party libraries, stopping cascade failures and allowing to improve resilience in complex distributed systems where the probability of failure is inevitable. |
| Edge Service | Are normally routers that provide authenticated access to faster, more efficient backbone and core networks. The trend is to make the edge device smart and the core device(s) "dumb and fast", so edge routers often include Quality of Service (QoS) and multi-service functions to manage different types of traffic |

## Appendix

NA