Logo

Description automatically generated

**API Integration Specification**

***Index***

*I.A-Administrator Authentication*, 2, 4

*I.B-User Authentication*, 4

*I.C-Errors*, 5

*I.D-Handling Errors*, 6

*I.E-Request Headers*, 9

*I.F-Uploading files*, 10

*I-Authentication*, 2

*II - Onboarding Services:*, 11

II- A - Onboarding Authentication: See I.A, 12

II- C - Submitting the documents, 12

II- D - Submitting the application, 13

II-B - Initiate the onboarding, 12

*III - Purchase Now Services*, 15, 18

III- A - Onboarding Shareholder: See section I.B, 15

III- B – List all the existing securities for a given customer, 15

III- C – Purchase Security, 16

III- C – Upload purchase orders, 16

*Introduction*, 1

*VI – Diagrams*, 18

*VI.A- Onboarding diagram*, 18

*VI.B- Purchase Security Diagram*, 19

***Introduction***

The Rialto API is intended for use by experienced developers; *it is not a low code / no code implementation*. The Rialto API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

You can use the Rialto API in the dev environment, which doesn't affect your live data. The url, secrets and credentials that you use to authenticate the request determines whether the request is live mode or test mode.

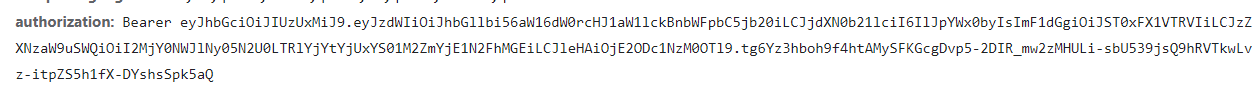
The Rialto API will be the same for every customer, but it will evolve through time as we release new versions and tailor functionality. These docs are customized to your version of the API.

***I-Authentication***

The Rialto API has two different levels of security: One for those services that can only be used by the customer´s administrators. In addition, on the other side, users that will authenticate with user and password can only access some other services.

Authentication to the API is performed via [HTTP Basic Auth](http://en.wikipedia.org/wiki/Basic_access_authentication). After you get the token through the administrator or user paths (see I.A and I.B), provide your token as the basic auth username value. You do not need to provide a password in every request.

In order to authenticate as a Bearer Mechanism:



**All API requests must be made over**[**HTTPS**](http://en.wikipedia.org/wiki/HTTP_Secure)**. Calls made over plain HTTP will fail.** API requests without authentication will also fail.

***I.A-Administrator Authentication***

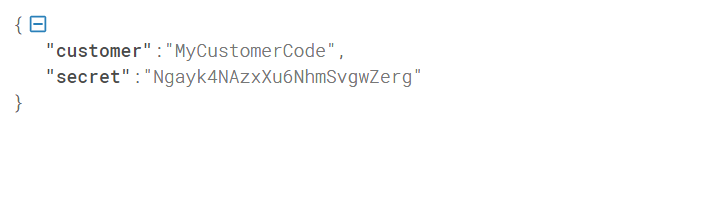
Interacting with the onboarding services does not require a traditional authentication mechanism, but you will have to use tokens the same way that was described in the previous paragraph.

In order to get this token, you will have to invoke a service sending the customer code and customer secret provided by Rialto. If you do not know your customer code and customer secret, please get in touch with Rialto support.

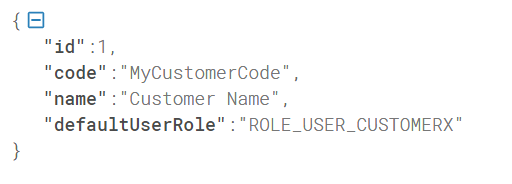
+ Action

POST ***<base\_url>*/secondarytrading/api/onboarding/authorize**

+ Request Body JSON

****

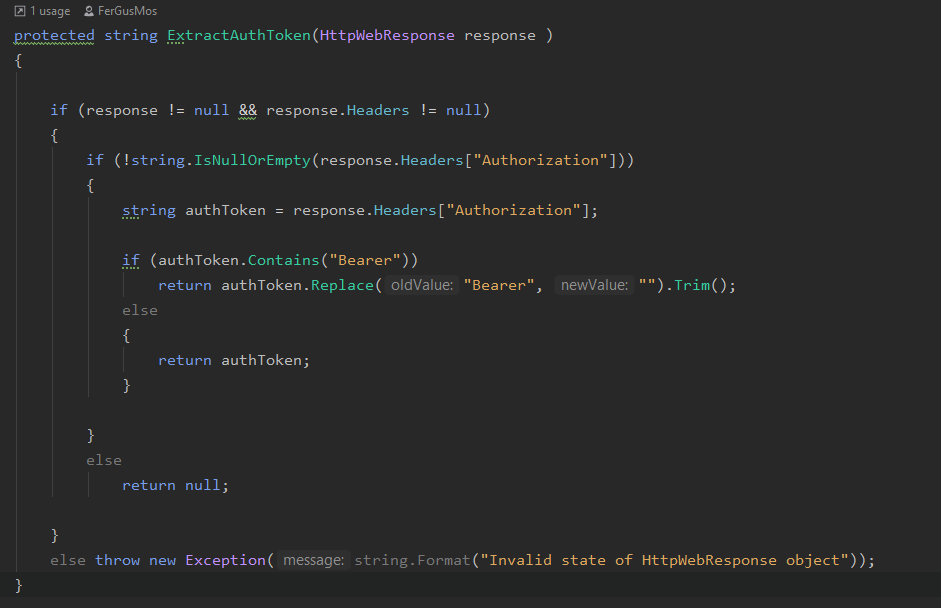
+ Successful response Body JSON

****

+ Token extraction

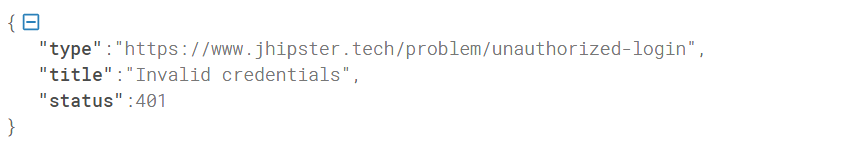
It is important to remark that the authorization token will not be part of the response Json, and it will have to be extracted from the Response Header, removing the Bearer prefix.

See the following example in C#



The previous is the authorization token that will have to be sent with every request.

+ Response with errors Body JSON (see ***section I.C***)

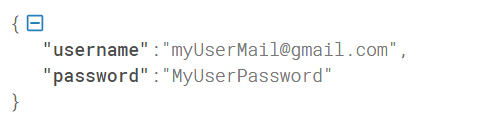
******

***I.B-User Authentication***

+ Action

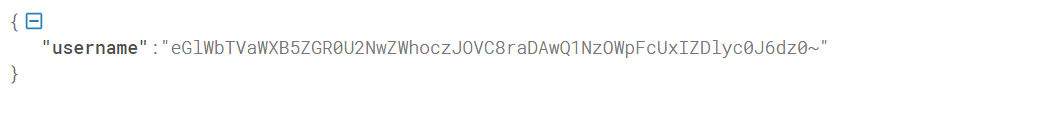
POST ***<base\_url>*/secondarytrading/api/authenticate?include-token=true**

+ Request Body JSON

******

Note: For all the error handling and request headers, see I.C and I.D

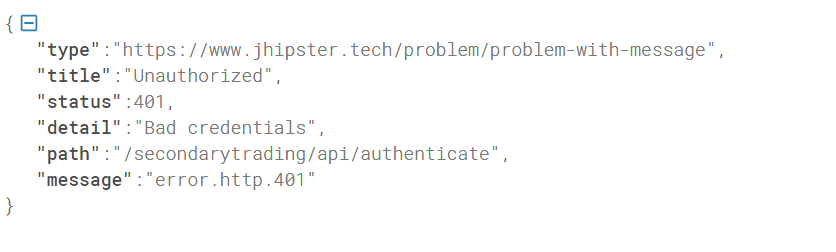
+ Successful response Body JSON

The previous is the *BCrypt* hash of the username used in the login process

+ Token extraction

Same as ***I.A-Administrator Authentication***

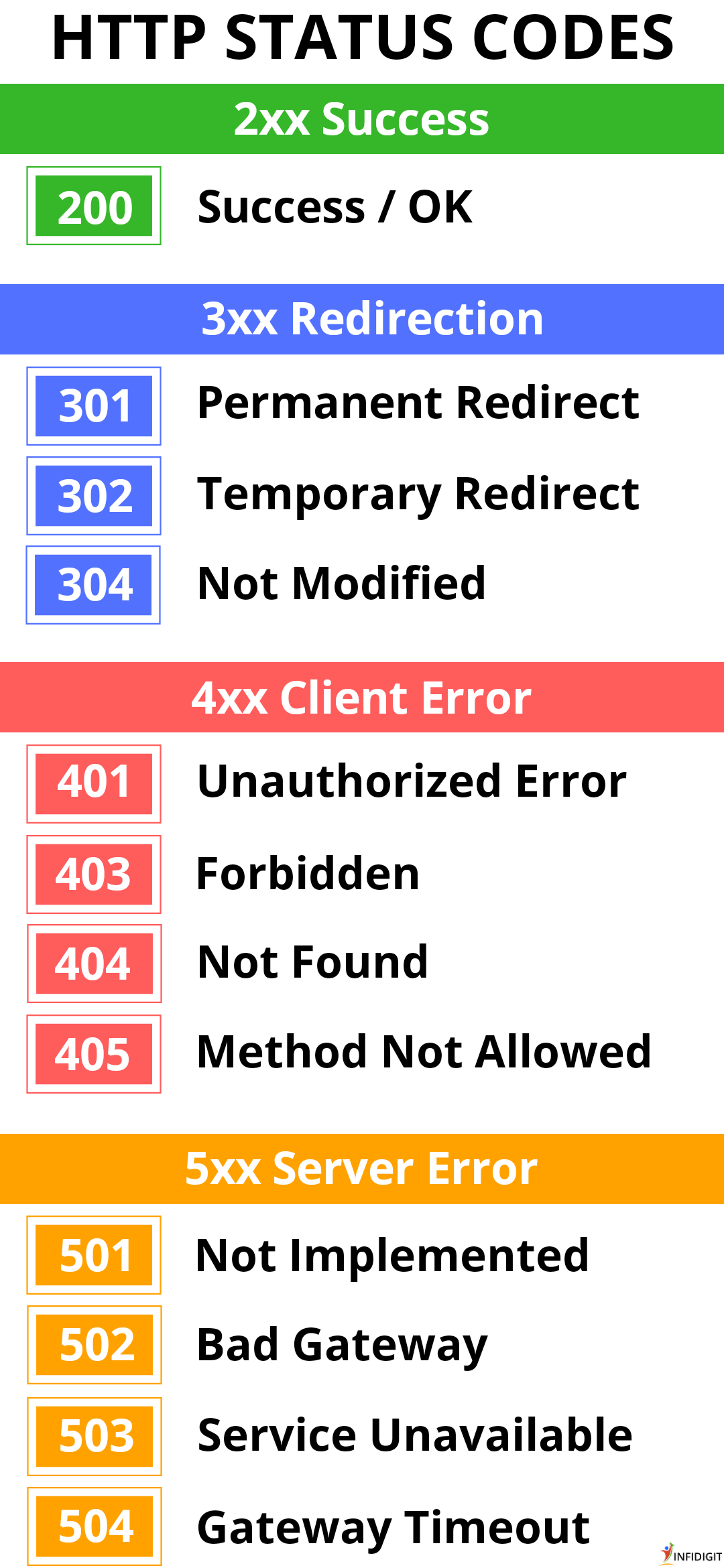
+ Response with errors Body JSON (see ***section I.C***)

******

***I.C-Errors***

Rialto uses conventional HTTP response codes to indicate the success or failure of an API request. In general: Codes in the 2xx range indicate success. Codes in the 4xx range indicate an error that failed given the information provided (e.g., a required parameter was omitted, a charge failed, etc.). Codes in the 5xx range indicate an error with Rialto´s servers (these are rare).

Some 4xx errors that could be handled programmatically (e.g., minimum investment violated) include an error code that briefly explains the error reported.



##### + Error response *attributes*

### type: The type of error returned.

### title: The title of the error.

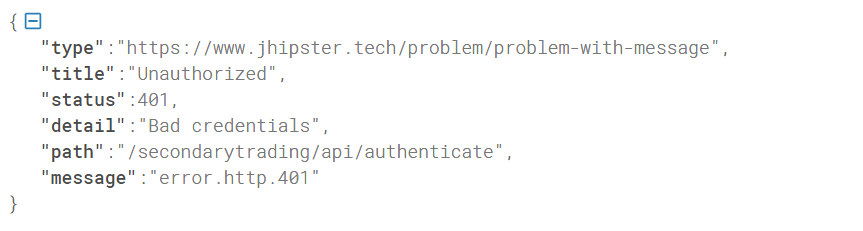
### status : HTTP Status code of the error.

### detail: Description of the error.

### path: The path of the service that threw the error.

### message: Extension of the detail message if necessary.

**An example of an error Json would look as follows**



***I.D-Handling Errors***

Our client libraries raise exceptions for many reasons, such as a failed charge, invalid parameters, authentication errors, and network unavailability. We recommend writing code that gracefully handles all possible API exceptions.

However, every service will return its own json describing the error structure that affects that specific service. Therefore, you will have two layers of errors that could be thrown:

+ An exception layer like the one that can be seen below, that will be thrown when the backend understands that there is a structural error that makes it necessary to communicate the error with a high-level exception

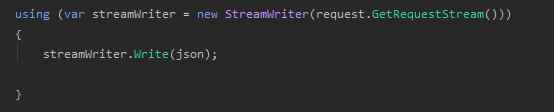
+ The response of every service, expressed as a Json that will inform any particular situation that might have taken place that it is important to report. This is handled this way, as the system communicates with several external vendors, whose faulty responses are returned to the service consumer who should be the responsible of properly handling these errors. So all these faulty responses will be described in every service body response structure.

When running a POST, you should implement the following error handling method. The example is written in C#:

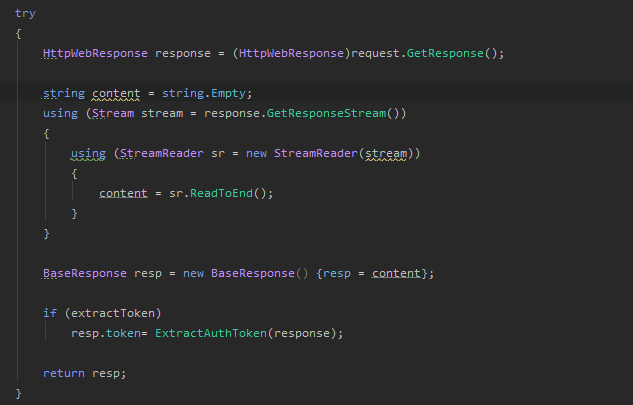
+ Setting up all the headers



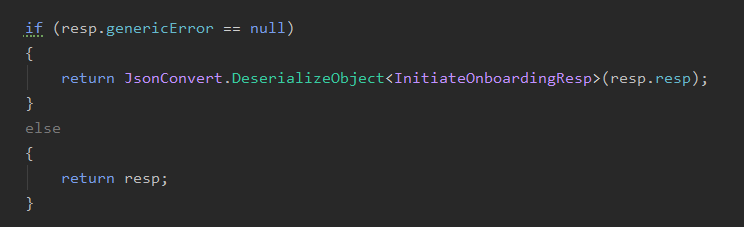
+ Sending the request



+ Process responses



+ Properly deserialize the response depending on the service invoked

******

**+ And if there are errors, properly deserialize the error message**

****

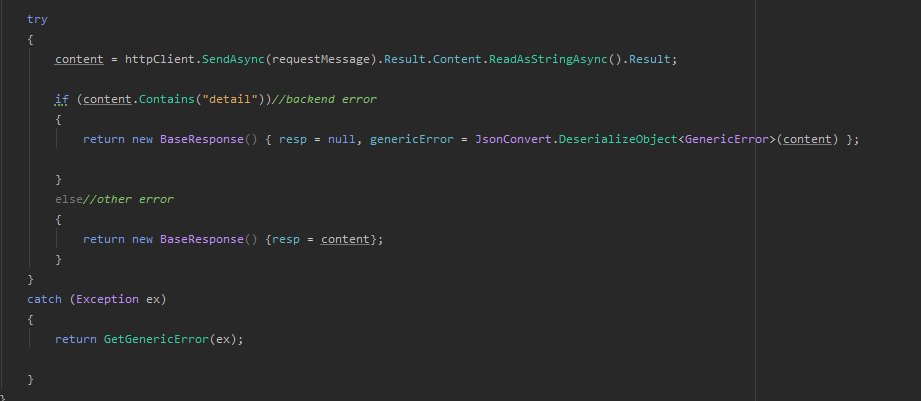
**Note: In Future version, *PaymentRailError* will be fully concentrated in the generic error format described in step I.C**

When running a GET, the logic is also similar. The example is written in C#:

+ Initialize the service



+ Properly handle the error messages



**Note: In Future version, *PaymentRailError* will be fully concentrated in the generic error format described in step I.C**

***I.E-Request Headers***

**HTTP header fields** are a list of strings sent and received by both the client program and server on every HTTP request and response. These headers are usually invisible to the end-user and are only processed or logged by the server and client applications. They define how information sent/received through the connection are encoded (as in Content-Encoding), the session verification and identification of the client (as in browser cookies, IP address, user-agent) or their anonymity thereof (VPN or proxy masking, user-agent spoofing), how the server should handle data, the age (the time it has resided in a shared cache) of the document being downloaded, amongst others.

**When running a POST, you should always send the following headers**

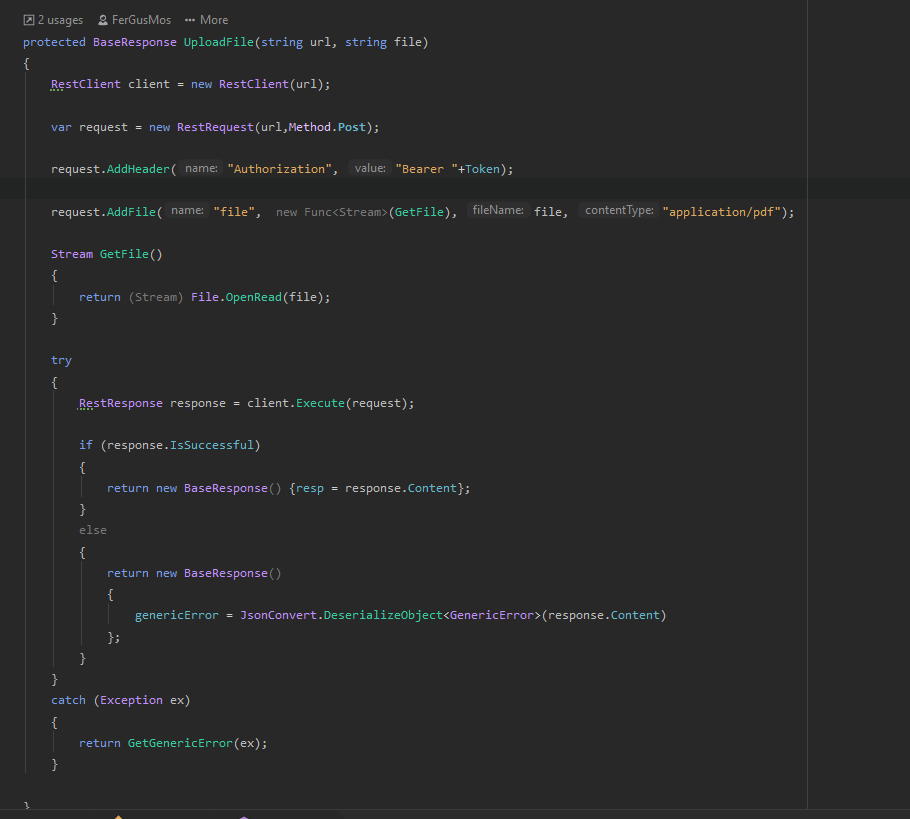


**When running a GET, you should always send the following headers**



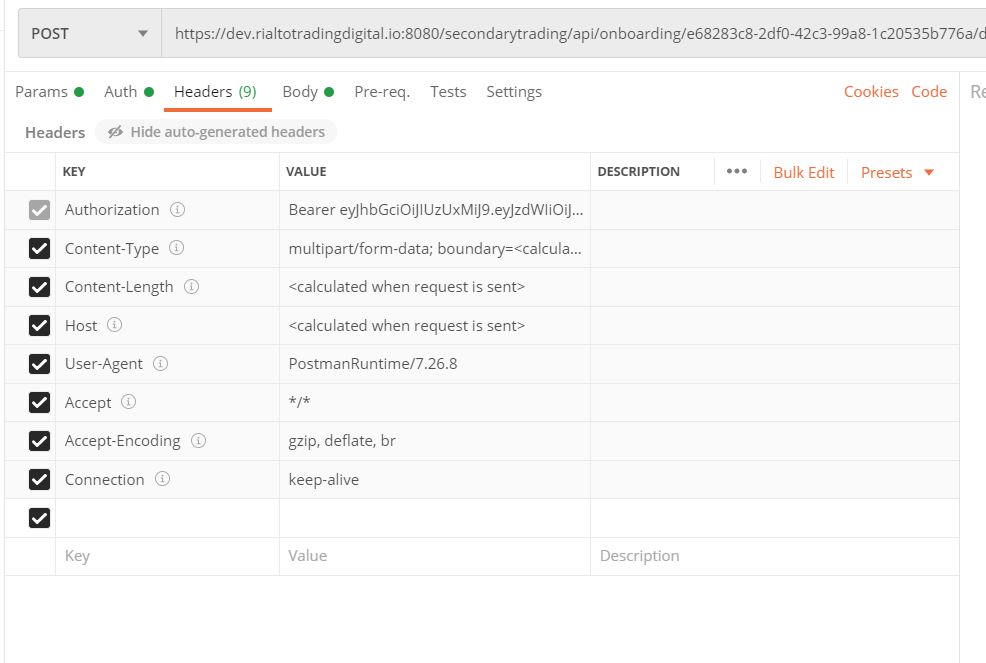
***I.F-Uploading files***

In several places of the onboarding and purchase flow, you will have to upload documents. Below you have a C# and Postman implementation of this upload mechanism.

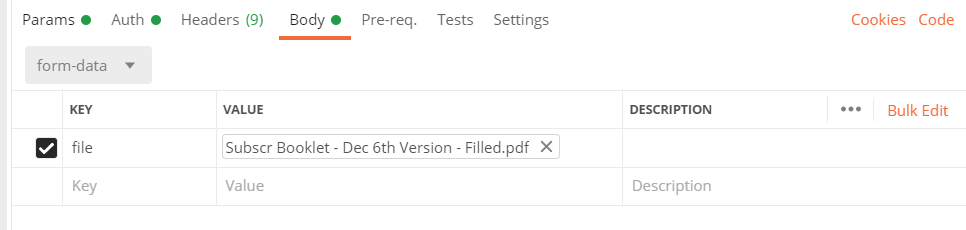
****

**For example, uploading the same file with Postman would look as follows:**

**+ Headers**

****

**+ Body**

****

***II - Onboarding Services:***

**Onboarding is the process when a new shareholder provides all the needed documentation to run the proper KYC/AML and enable that user in the platform creating its users and sending all the links so that he can reset his password.**

**At the time of writing this document, everything will be handled through a 4-step process that invokes the following 4 services**

**II- A - Onboarding Authentication: See I.A**

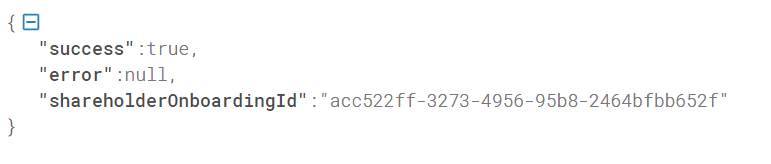
**II-B - Initiate the onboarding**

***+*** *Action*

POST ***<base\_url>*/secondarytrading/api/management/onSecondaryMarketInitiateOnboarding/ *{CustomerId}***

*+ Body JSON must be empty*

**+ Successful response**

****

**Where the *shareholderOnboardingId* is the external onboarding id that will have to be used wherever you find the tag *{OnboardingId}***

**If there is an error, the *success* attribute will be false, and the error will describe the existing error. For example, if you provide a wrong or not recognized *{CustomerId}* you will receive a rejection message.**

**II- C - Submitting the documents**

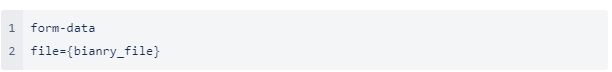
**+** *Action*

POST ***<base\_url>/*secondarytrading/api/onboarding/*{OnboardingId}*/document?type=*{Type}***

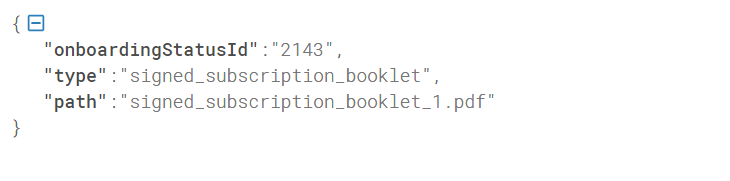
*+ Parameters*

* ***{OnboardingId}*** is the OnboardingId received when invoking ***onSecondaryMarketInitiateOnboarding***
* ***{Type}*** can be
  + signed\_subscription\_booklet
  + signed\_subscription\_agreement
  + signed\_ats\_agreement
  + W\_9\_form
  + verification\_letter\_user

*+ Headers*



*+ Successful response*



Where the ***onboardingStatusId*** field is the internal onboarding id.

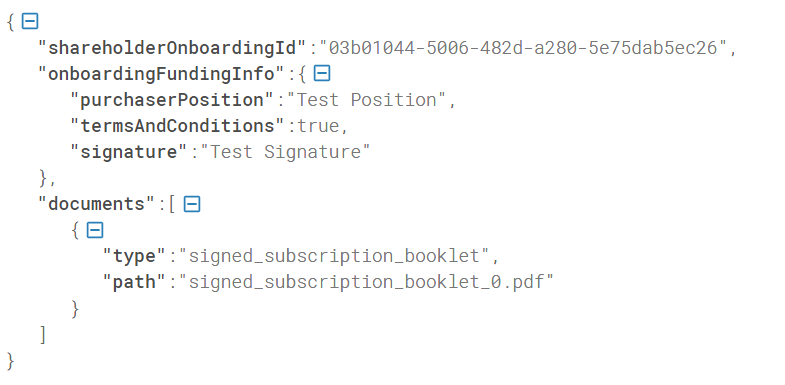
*+ Response with errors: see section* ***I.C-Errors***

**II- D - Submitting the application**

**+** *Action*

POST ***<base\_url>/*secondarytrading/api/management/onSecondaryMarketApplicationRequest2**

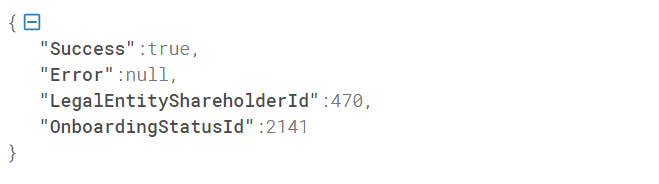
**+** *Body JSON should be the following*



Where

* ***shareholderOnboardingId*** is the ***Onboarding Id*** received when invoking ***onSecondaryMarketInitiateOnboarding***
* ***signature***: Free string
* ***purchaserPosition*** : Free string
* ***termsAndConditions***: true

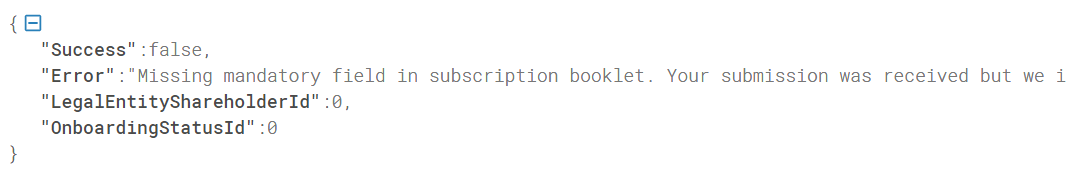
**+ Successful** Response JSON



Where

* ***OnboardingStatusId*** : Is the *internal* *Id* of the previously initiated onboarding in step II.B. The numerical Id and the id received in step II.B are mutually exclusive. One is internal and the other is external.

**+** Response w/Errors JSON



After all the previous steps have been properly executed, all the shareholders mentioned in the key authorized section of the subscription booklet will receive welcome emails inviting them to go through all the steps of the AML/KYC process.

After everything is approved and the administrator has approved the application, these key authorized persons will receive an email inviting them to reset their password. They will reset their password and they will be able to log in to the Purchase Now platform.

***III - Purchase Now Services***

**III- A - Onboarding Shareholder: See section I.B**

**III- B – List all the existing securities for a given customer**

**+** *Action*

GET <base\_url>/secondarytrading/api/primary-company?customer=***{CustomerId}***

+ Where

***{CustomerId}:*** Returns all the tokens/securities that belong to that to that customer (Ex: **Masterworks**)

+ **Successful** Response Json

It will be an array of the following structures



Where

* + ***Id***: Internal id of the security
  + ***companyLegalName***: Descriptive name of the security
  + ***email***: Email of the responsible of that security
  + ***comments***: Descriptive comment of the security
  + ***securityDescription***: Second level description of the traded security
  + ***stock price***: Initial or Purchase price of the security
  + ***minimumInvestment***: Minimum amount of units to buy of the security
  + ***symbol***: Security symbol that will identify the security in the Rialto platform
  + ***issuerLogo***: Logo assigned to that security
  + ***regulation***: Regulation assigned to that security (if available)

**+** Response w/Errors Json

An empty json will be returned if no securities are found for the ***{CustomerId}*** provided

**III- C – Upload purchase orders**

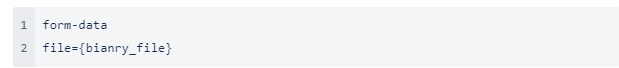
**+** *Action*

POST ***<base\_url>/*secondarytrading/api/primary-purchase/company/*{Symbol}*/document**

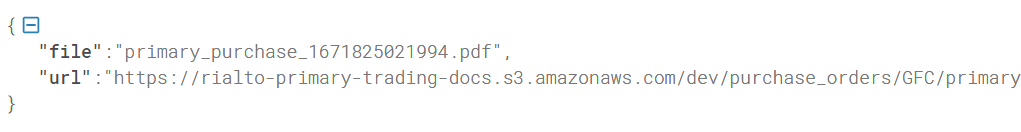
+ Where

***{Symbol***} is the token/security symbol

+ Header (see I.F)



+ **Successful** Response JSON



**III- C – Purchase Security**

**+** *Action*

POST ***<base\_url>/secondarytrading/api/primary-purchase***

**+** *Body JSON should be the following*

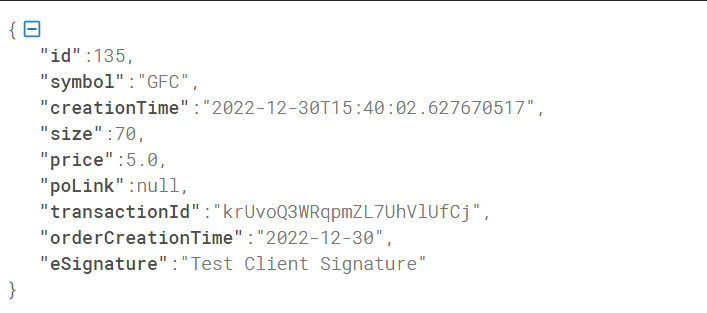


Where

* + ***eSignature***: Non mandatory signature of the purchaser
  + ***orderCreationTime***: Full date time when the order creation should be accounted. Format ***yyyy-MM-ddThh:mm:ss***
  + ***symbol***: Symbol of the security to purchase
  + ***Size***: number of shares/tokens/units to purchase

The purchaser user and shareholders will be extracted from the authentication token.

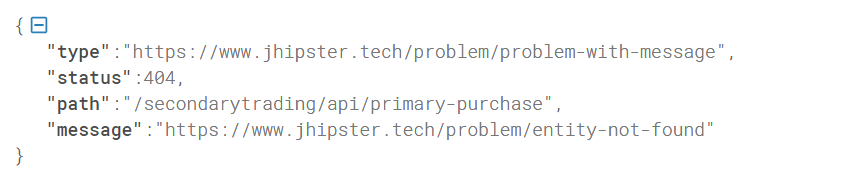
+ **Successful** Response JSON

****

Where

* + ***id***: internal Rialto id of the purchase
  + ***creationTime***: Full date time when the purchase was executed. Format ***yyyy-MM-ddThh:mm:ss***
  + ***symbol***: Symbol of the security purchased
  + ***Size***: number of shares/tokens/units purchased
  + ***Price***: Price paid pair share/token/unit purchased
  + ***transactionId***: External Rialto id of the purchase
  + ***orderCreationTime***: Echoes value received in the purchase
  + ***eSignature***: Echoes the signature received in the purchase

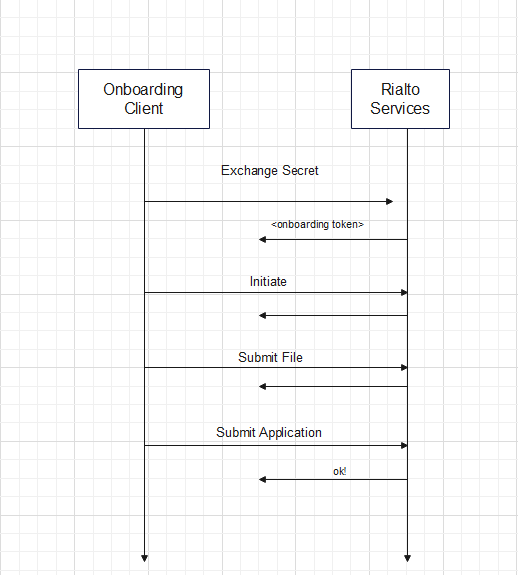
**+** Response w/Errors Json



***VI – Diagrams***

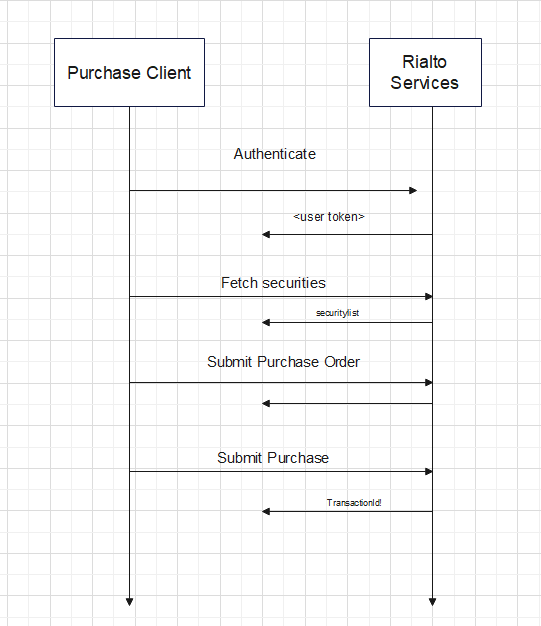
***VI.A- Onboarding diagram***

**The following diagram describes the whole onboarding sequence in a single picture**



***VI.B- Purchase Security Diagram***

**The following diagram describes the purchase security process in a single picture:**



**VII: FAQ**

***VII.I- Password change***

**In order to change a user’s password, please contact the following email address:**

[support@rialtomarkets.com](mailto:support@rialtomarkets.com)

***VII.II- URL List***

**The following is the url list of the different Rialto environments (see *base\_url* tag)**

**+ development**

<https://secondary-trading-service-dev01.services.rialtomarkets.io>

**+ staging**

<https://secondary-trading-service-stage01.services.rialtomarkets.io>