**Introduction**

The purpose of this document is to describe all the technical, functional and nonfunctional features involved in the ***Rialto ATS Secondary Market*** platform. It is dedicated to Rialto staff, technical and non- technical users, and it expects to have in an easy and straight language a full description about all the Rialto’s functionalities, and all the things that are worth to know to properly manage and set the Rialto ATS platform.

This document does not pretend to be technical, but a few technical concepts will be described, like tables, services, technologies, deep enough so that a non-technical users can understand the main concepts that have to be known, but without digging into too much detail that would be beyond the scope of this document. However a certain level of technical knowledge must be necessary in order to properly deal with those concepts.

It is worth to mention that throughout this document you might find some sentences in red color. This sentences are written in this color as a convention to indicate that at the time of writing this document, further research had to be taken in that given subject.

It is also worth to mention that some other very specific functionalities, like all the technical details in the integration with the transfer agents, will be described in separate documents like ***Rialto KCX Solidus integration - Functional specs v1.x.***

Another important detail to mention is that at the time of writing this document, some of the most important settings are managed through tables, until the proper management functionality is built. **This kind of technical references will be described in blue**, and will later be updated to include the management functionality developed, once it is ready.

After version 1.4 of this document, a whole new module describing all the Primary Market functionalities will be included in this document.

**I - Entities Structures**

The ATS platform pretends to be a platform that will allow different shareholders to trade private equity shares that same way that they would do in all the known public markets.

So the platform will have to deal with the same concepts that anyone used to the capital markets has usually dealt with:

* Companies
  + They will represent that the organization that will have one or many of its securities offered in the exchange
* Securities
  + One company can offer several kinds of assets like equity, fixed income and derivatives. At the time of writing this document, most of the securities traded are regular shares.
* Shareholders
  + They are the users of the platform which can use it to
    - Have access to publicly available information about the different companies or securities
    - Monitor their holdings and bank account information
    - Trade the different securities listed in the platform.
    - Request the funding and redemptions of their trading accounts
* Trading
  + This refers to
    - Orders
    - CLOB order books
    - Matching
    - Settlement
    - Etc.
* Administrators
  + They are the rialto operators responsible of
    - Managing the different companies and securities populating the different media, links, documents, management teams and all the relevant information that will be used by the shareholders to be properly informed about the different securities available for investment
    - Managing all the trades settlement process, dealing with the different steps when clearing trades, from moving funds to communicating with the transfer agents to move the shares to the proper destination
    - Managing all the default or problematic scenarios
    - Manage all the platform settings
    - Manage and monitor all the new shareholders onboarding process
    - **<Other administrator functionalities to be added>**
* Users:
  + As with most of the available applications, the ATS has 2 basic levels of users (See **II.A – Users)**
    - ***Administrators***: When logged in they will have access to a backend site, which will allow them to manage all the administrative entities mentioned before. This is
      * Manage shareholders
      * Manage Securities
      * Control the workflow of transfer fund settlements
      * Control the workflow of the trade settlements
    - ***Investors***: These will be the shareholder’s users. This will have access to the trading site, which is the site that will give them all the needed tools to properly analyze each company, have all the market activity information and create buy and sell orders.

**I.A - Shareholders**

As was mentioned before, the ***Shareholders (Investors)*** are one of the main entities in the platform as they are the users that the ATS is addressed to.

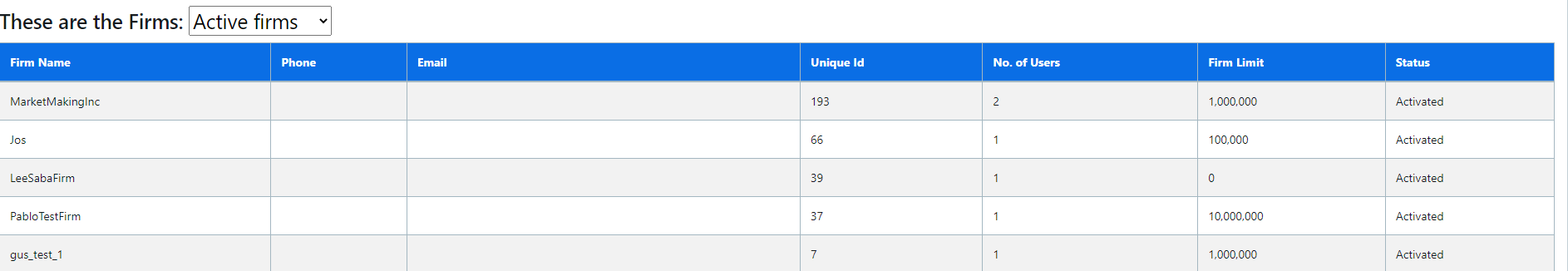
They will be the users that will give life to the platform using all the available functionalities, to inform themselves about a given company, see all the market activity, fund their accounts and trade their holdings.

Managing shareholders

Once having accessed as an ***administrator*** to the platform, the following button will allow the admin to see all the available shareholders.

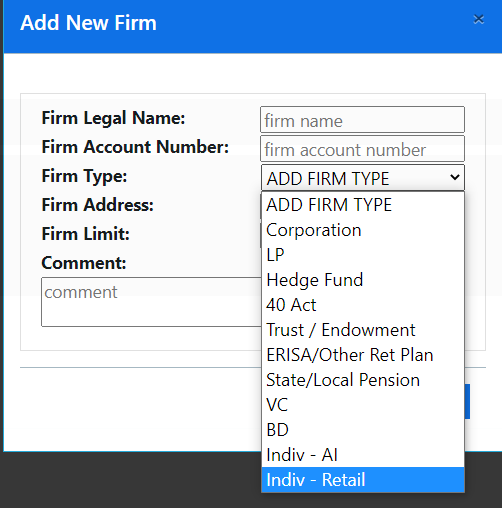


This will allow the administrator to have access to all the ***shareholders*** *and* ***administrators*** registered in the platform L:



It is worth to mention that there is kind of an exotic way to deal with physical persons at the time of writing this document. All ***shareholders*** will be managed as ***firms*** until a more friendly interface is implemented that allows the ***administrator*** to differentiate *legal entities* from *physical persons*.

So if a new physical person had to be added to the platform, this should be done creating a firm and then assigning this “*firm*” an “***Individual Retail***” type

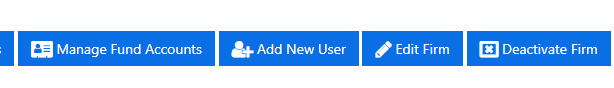


Anyway, the safest and most robust way to add a new ***shareholder*** to the platform will be though the onboarding process that will be later fully described. This process allows to run a full KYC process through the ***Solidus*** platform, create credentials and bank accounts in the Piermont bank. **This process will be fully described later in this document**.

A ***shareholder***, as a representation of a *physical or legal entity*, can have several users used to log into the trading platform.

In order to manage a given ***shareholder’s*** users, click the selected “firm” and that will display all the users enabled.

Once in this screen you will be able to:



* Create new users
* Edit the “firm” (*physical or legal entity* information)
* Deactivate the “firm” ***(shareholder)***
* Mange the ***shareholder*** accounts (see “*Managing Fund Accounts*”)

Managing Fund Accounts

In the ATS platform there will be three level of accounts needed for trading

1. The ***fund account***: These are virtual accounts of every *shareholder.* They don’t have a physical or real representation as they don’t hold real money. However they will be linked to real ***Piermont******accounts*** *(see #2)*. So, these accounts will have a FIAT balance that will be synchronized to a real ***Piermont account*** which is, as it is explained in dot #2, the real account from which the funds are going to be transferred or received when clearing the trades. These accounts are managed through table **fund\_accounts**.
2. The ***Piermont account*** which will be linked to a ***fund account*** mentioned in the previous section #1. This account will be created and linked automatically through the onboarding process. **At the time of writing this document there is not possibility to manually link a *Piermont account* to a *fund account* and this can only be done automatically in the onboarding process**. So basically, every time the user picks a ***fund account*** when creating an order, it will be the ***Piermont account*** the real account from which funds will be extracted or deposited when clearing a trade. **If the user picks a *fund account* that is not linked to a Piermont account, this is not restricted at the time of writing this document, but it will be the Rialto operator the one responsible of managing this scenario at the time of clearing such trade.** These accounts and its linkage to the fund accounts is managed through table **fund\_account\_details**.
3. A third type of account that is managed in the ATS platform is a **Plaid account**. A ***shareholder*** can have as many external accounts linked to his ATS user and this will mostly used to fund his Piermont account through the ATS. **At the time of writing this document this linkage is done automatically through the onboarding process in the Solidus platform. This platform will provide what is called a token, which is a string that identifies an authenticated user in an external bank (ex: Bank of America). Later, this token will be stored in the *plaid\_credentials* table and it can be used to access the shareholder’s private bank account number and routing number, necessary to create the NACHA files used to transfer funds from the external bank accounts. However, it might be a good idea to give the shareholder in the future to possibility to link his external bank account in the ATS.**

So in summary

* When creating trades, the ***shareholder*** will pick a ***fund account*** (#1) and that will provide the ATS all the **Piermont** account’s necessary information to clear that trade in case there is a match.
* When funding is requested, the system will use the **Plaid** token populated in the onboarding process, to gather all the necessary information to send to **Piermont** and have the extraction and deposit properly executed.

**I.B- Transfer Agents**

Before saying anything is worth to mention that there is already a transfer agent onboarding document that describes with much more details all the different interactions that take place with the different transfer agents.

* **Transfer Agent onboarding - Reference Manual v1.x**

Basically, a transfer agent is a company responsible of keeping a ledger where it stores who owns what. So the interaction with the transfer agents will have the following contact points

* + Incoming
    - When onboarding a new shareholder that is coming from the transfer agent
  + Outgoing
    - When trading
      * Putting shares on hold and release them when the hold is no longer needed
    - When clearing trades
      * Requesting to transfer the shares
    - When synchronizing positions

As can be seen, most of the interaction start from the ATS and request something to the transfer agent. In this context the most important tables that belong to a given transfer agent interaction, will have a 3 letters prefix indicating what’s the transfer agent referred by that table. Then, that table will have all the fields that are needed for that specific functionality.

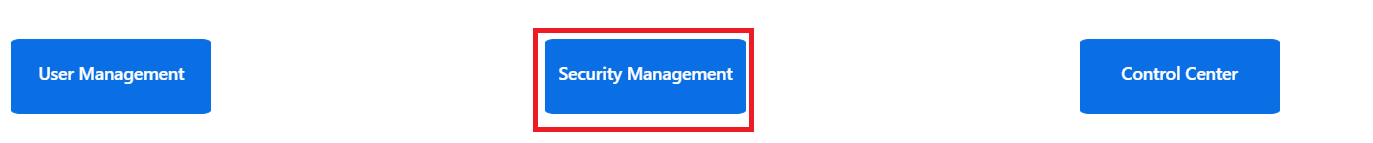
For example, the transfer agent ***KoreConX*** will have the following tables

* **kcx\_kore\_security**: For every security , it stores all the Kore Con X ids of that security (called kore chain ids or kore security ids in the Kore Con X world). This table will have to be read every time we want to make a POST/GET request referring to a specific security
* **kcx\_kore\_shareholder\_ids**: For every shareholder, it stores all the ***Kore Con X*** ids of that shareholder (called kore chain ids or kore shareholders ids in the **Kore Con** X world). This table will have to be read every time we want to make a POST/GET request that affects in some way, a given ***Kore Con X*** investor
* **kcx\_connection\_settings**: This table stores all the urls, keys and user/passwords of all the Kore Con X servers, needed to connect and properly authenticate in those servers.

But remember, the previous is just an example. New transfer agents, will have different requirements and logic, and they will require the same and maybe more tables. But always remember to respect the 3 letters prefix for every transfer agent, so that we know what transfer agent every table refers to.

**I.C - Securities**

As was mentioned in the introduction, the primary purpose of the ATS secondary market is to trade shares and also other asset classes. All these different securities can me managed after clicking in the following button of the administrator UI:

****

**It is a fact that a company can have multiple securities (shares, preferred shares, bonds, warrants, etc.) but in the current version of the platform we will create different securities as a whole. This means, if a company wanted to have 2 securities listed for trading, they would have to create two different securities in the previous screen, which would be unrelated one from the others. In summary, at the time of writing this document, the ATS manages securities, not companies. It might be a desirable feature for future versions to group all the related securities under the company they belong to.**

Creating a security is probably the most flexible and dynamic process that a Rialto **administrator** can go though. This is because this screen allows the ***Rialto administrator*** to manage a lot of different aspects of that security, and this includes

* All his financial attributes like *Symbol, CUSIP, notional, coupons*, etc.
* All his visual representation like *links, website, media, documents,* etc.
* All the corporate valuable information like *research reports or the management team*
* All the different keys to the different *transfer agents*

So it is important to know that everything that is implemented in this screen, will have an impact in the way that security is not only traded but also presented to the final user. The ***shareholder UI*** is very flexible and it is here where all the actions taken in the ***Security Management*** section will have effect.

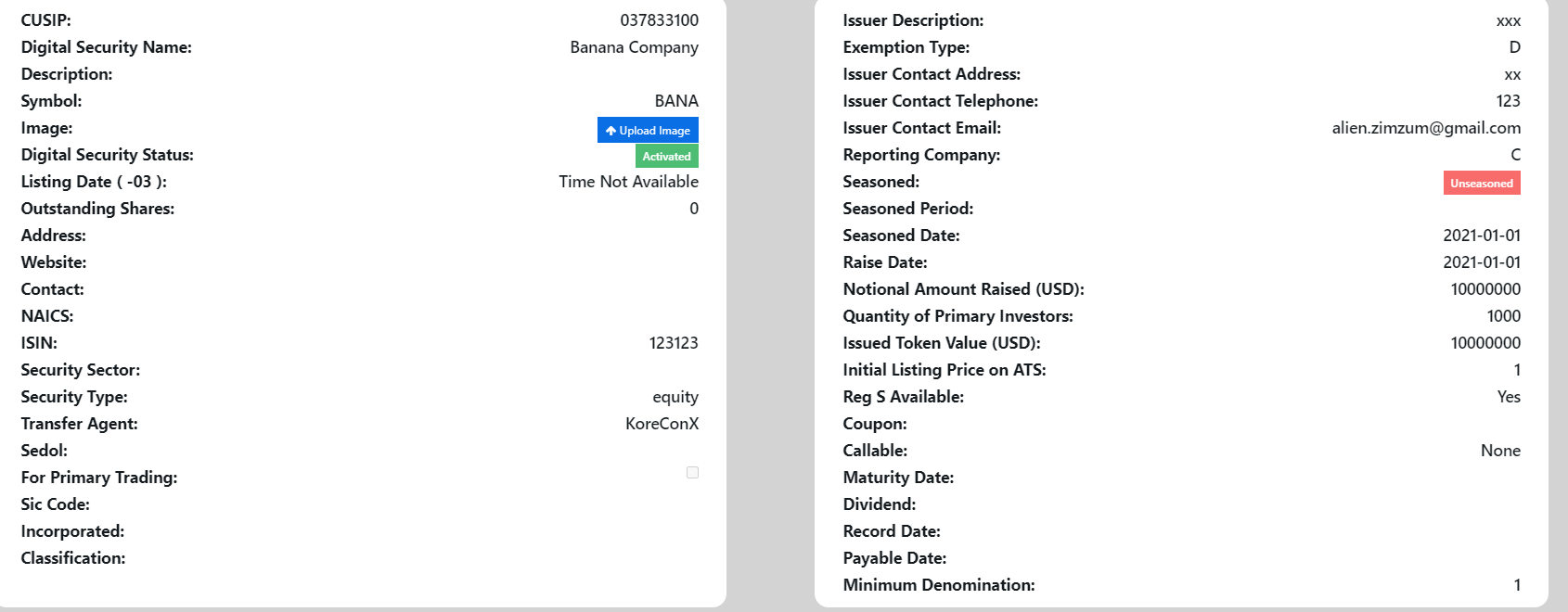
The following are the most important screens to manage a security:

Create or Edit a Security main parameters

** **

Here we will have a screen that will allow us to populate a new/existing security main attributes like ***symbol, CUSIP, coupons, notional***, etc.

The Digital Security entity will be managed through the ***fund\_security*** table.



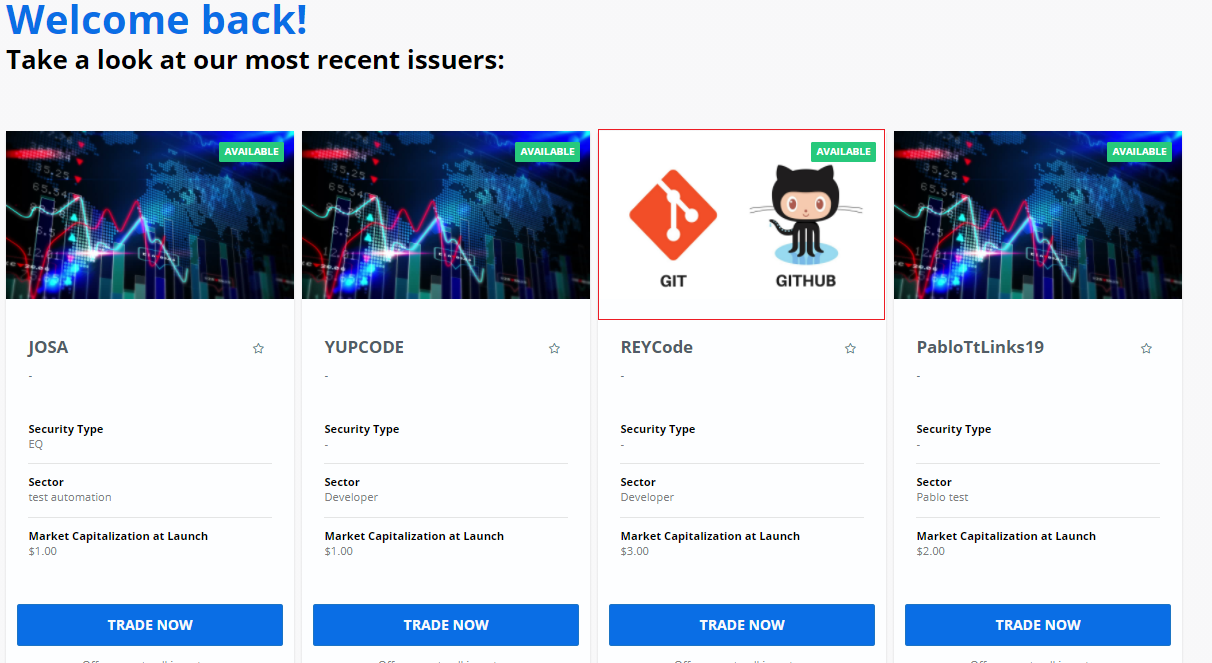
Some of these attributes, like the symbol, will be actively used through the security life span, others just have an informative purpose at the time of writing this document.

Assign a security its main image

Clicking in the following link, you will be able to upload the main security image.

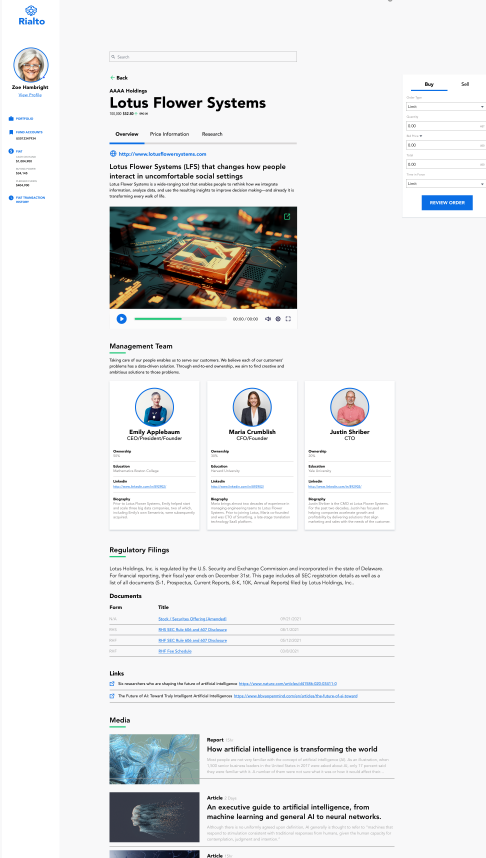


This is the image that will be displayed in the following security panel in the ***shareholder UI***



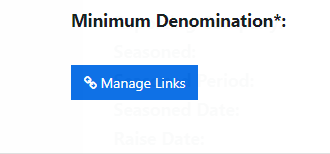
Link management

Whenever a security is displayed in the ***Shareholder’s*** UI, different component will be displayed that will reflect the security most important facts, news and market activities.

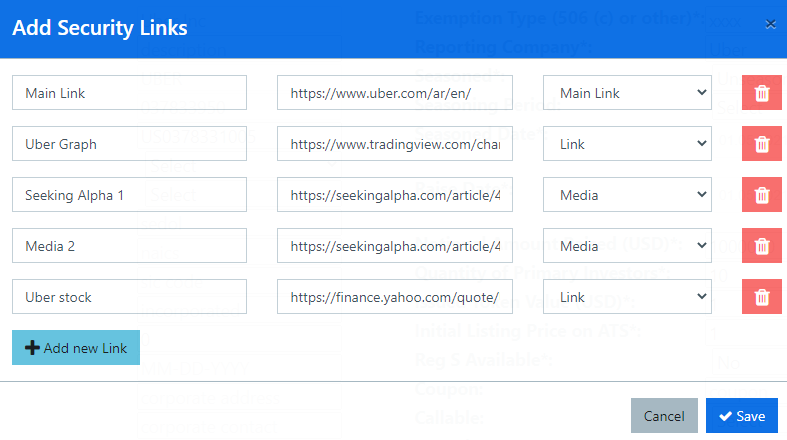


The reality is that most of the previous components, are displaying web pages links.

All these links are managed in the administrator´s security management UI. If you look at the bottom, you will see the following link



When clicked a new popup will show up which looks as follows:

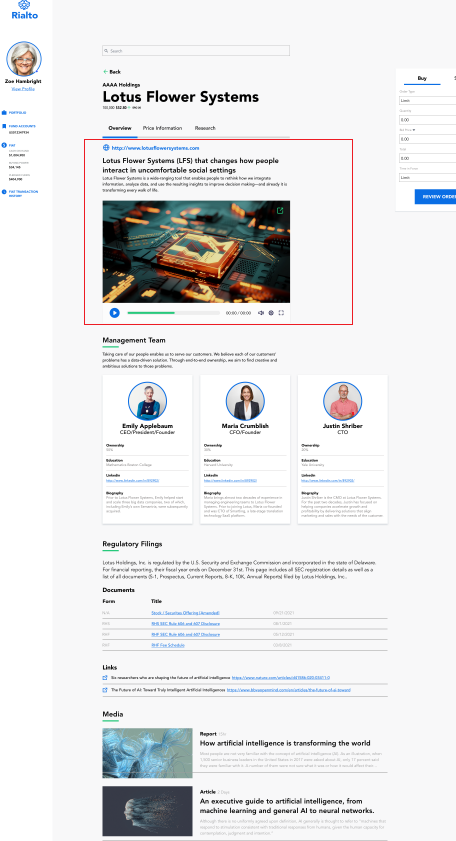


As you can see here you have a list of server “links” where you have a **main description** textbox, the **link** textbox and a third combo which has the following options:

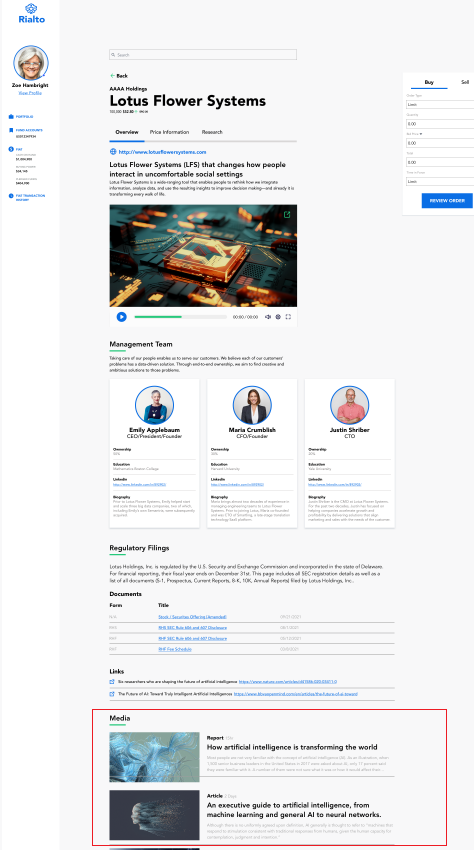


Whatever you pick in this combo will define where the referred link is displayed.

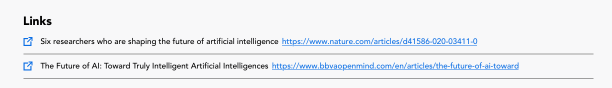
The ***Main Link*** will be displayed in the following region



The ***Media Link*** with the title and images, will automatically be displayed in the following region



Moreover, the regular links will be displayed as follows:



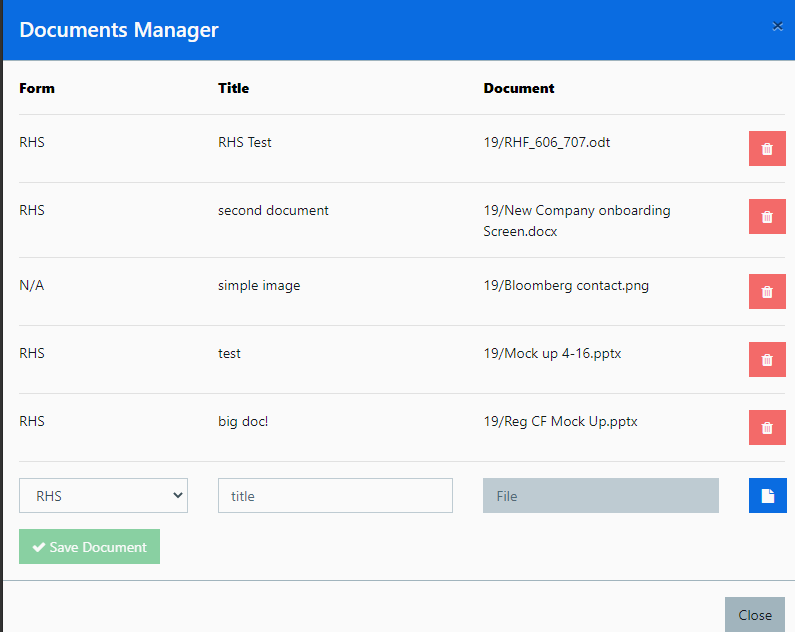
Document management

The very same way that we could manage the security links, we can manage all the different **documents** of that security.

If you look above you have the following link:



This link will open the following popup



And all these **documents** will be displayed in the following component

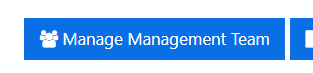


Remember that all these documents are public documents. This means that you should only upload here documents that will be visible and ready to be read by the different **shareholders**.

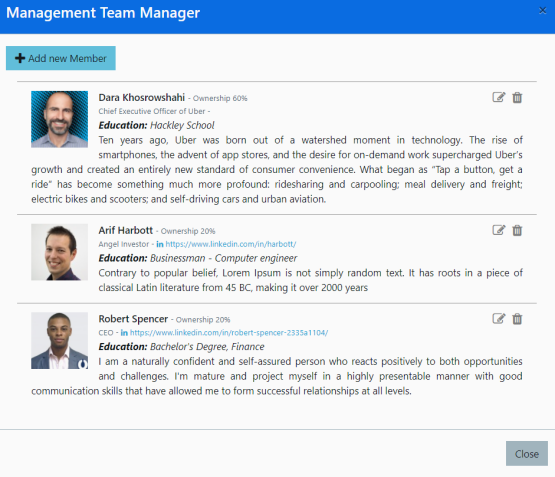
Management team population

The very same way that we could manage the security links, we can manage all the different **management team members** of that security/company.

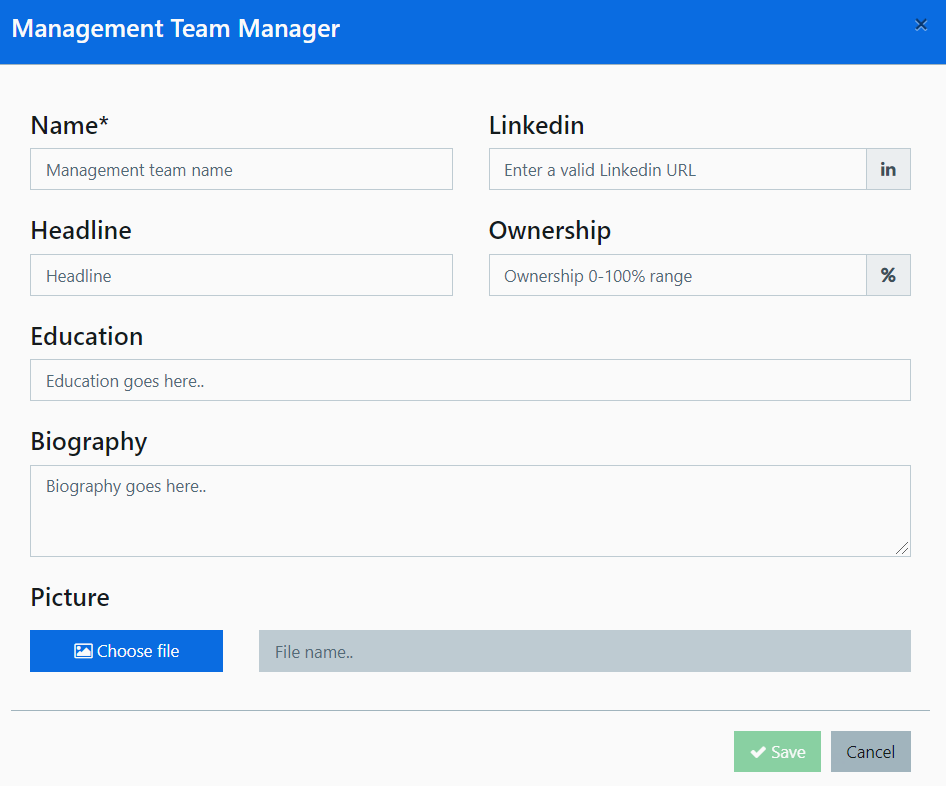
Once inside a security dashboard, clicking the following



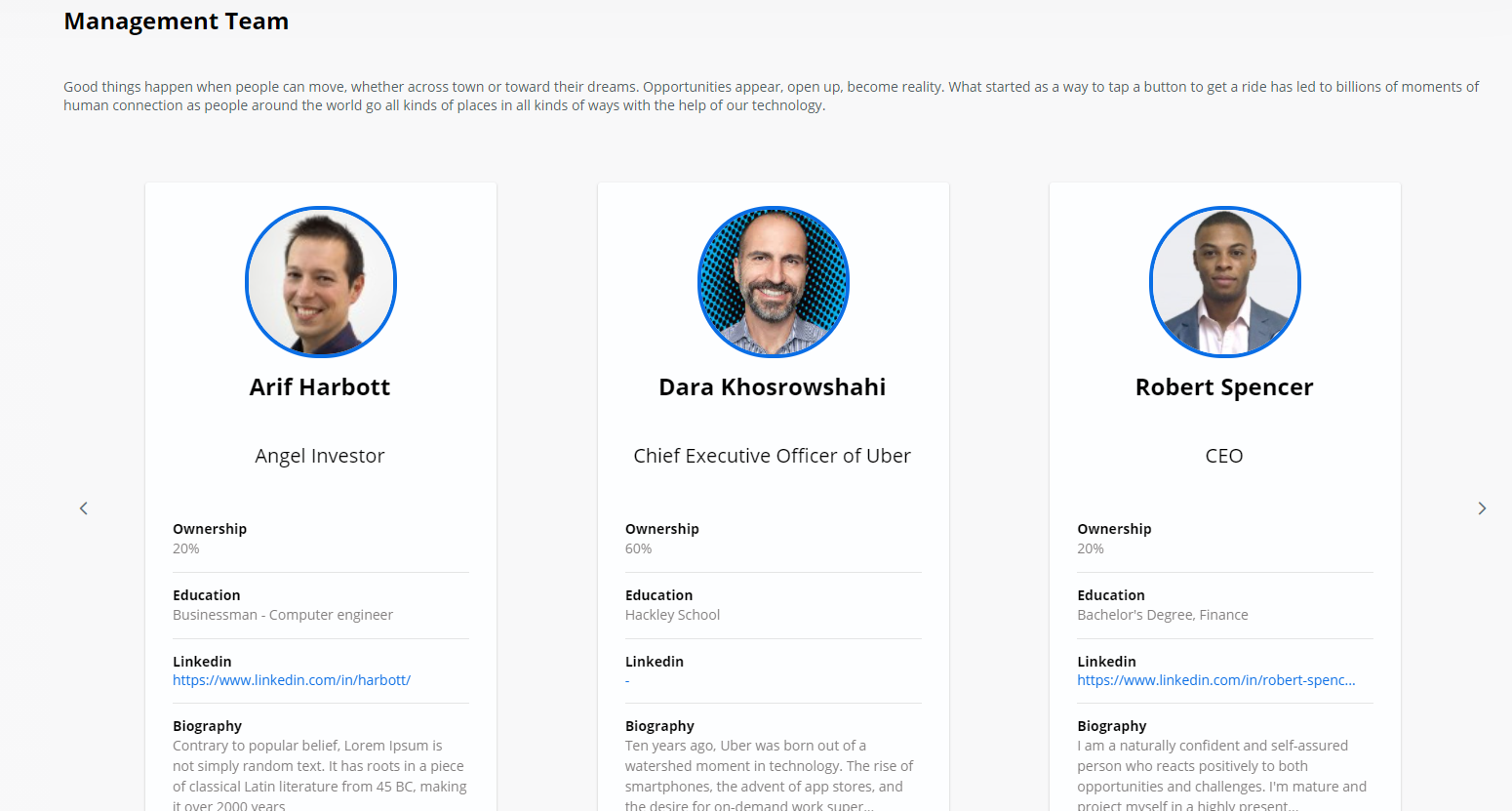
You will see all the previously loaded management team members



Clicking the Add new Member button, you can populate all the following fields



All the populated team members, will be later displayed in the shareholder’s UI, *Overview Tab*



**I.D – Trade Settlement**

The settlement process is one of the most important in the operations of a stock exchange. Basically it will be one of the most complex processes of all, where a proper instrumentation (logging), a smooth, performant and intuitive interface have to be implemented to have solid and robust settlement.

**At the time of writing this document, most of this workflow is under design**. **The document will be updated, but many changes can arise and this document will have to be updated accordingly**. The special complexity lies in the fact that it is in this specific functionality that the ATS will be communicating through different steps with different actors (banking institutions, transfer agents, etc.) and it will be the ATS the one responsible of granting the consistency of all these interactions. The failure of a specific actor, should not propagate to an inconsistent state or to a different actor.

Matching trades

One of the newest features of the ATS is to have a central limit order book (CLOB) that replaces the order matching with discrete executions.

Basically, every time a new buy(sell) order is created, there will be an engine that will look for any previously existing trades with equal or lower (higher) price. If that happens, we will have one or several trades. This algorithm will go one recursively until no more trades can be matches.

In case that 2 orders have the same price, the one that arrived first will always have a higher priority in this matching process.

**If there is a match, an email will be sent to both participants and a new record will be created in table clob\_trade.**

For more information about the specific details of the behavior of a CLOB, you can refer to the following link:

**https://www.investopedia.com/terms/l/limitorderbook.asp**

Workflow Commands

When clearing the trades, you will see the following commands that will allow implementing all the needed actions to finally have the money and the shares transferred.



Fiat Settlement



This button has to be pressed to create the NACHA file that will be sent to Piermont to transfer the fiat from buyer to seller.

After being pressed the trade will be marked as ***Bank Info Exported***.

Fiat Transfer Confirmation



This button has to be pressed to confirm that the fiat has been received by the seller. At the time of writing this lines it is being evaluated if this can be done automatically.

After being pressed the trade will be marked as ***Fiat Transferred***.

Shares Transfer Request



This button has to be pressed to initiate the shares transfer with the transfer agent. Some transfer agents have a synchronous transfer, other have an asynchronous one.

The important thing to mention is that the transfer request will be started with this button. Then we can go to the next step to confirm if the trades could be actually transferred.

After being pressed the trade will be marked as ***Shares Transferred Requested***, unless we have a synchronous confirmation, in which case, the trade will be marked as **Trade Cleared.**

Shares Transfer Confirmation



This utility of this button will depend in how exactly the transfer agent is confirming that the shares transfer is successful

* If the shares transfer is asynchronous, when pressing this button, depending on the transfer agent
  + The shares will be marked as transferred
  + The proper service will be called to confirm that the shares have been transferred
* If the transfer agent is synchronous, this button will be automatically pressed, after pressing the “Shares Transfer Request” button

At the time of writing this document, in Kore Con X, we are waiting for a “confirm shares transferred” service to be called.

After being pressed the trade will be marked as ***Trade Cleared***.

Default Scenarios

**<TO COMPLETE: Resarch Default Scenarios>**

**<TO COMPLETE: Order´s expiration and cancellation>**

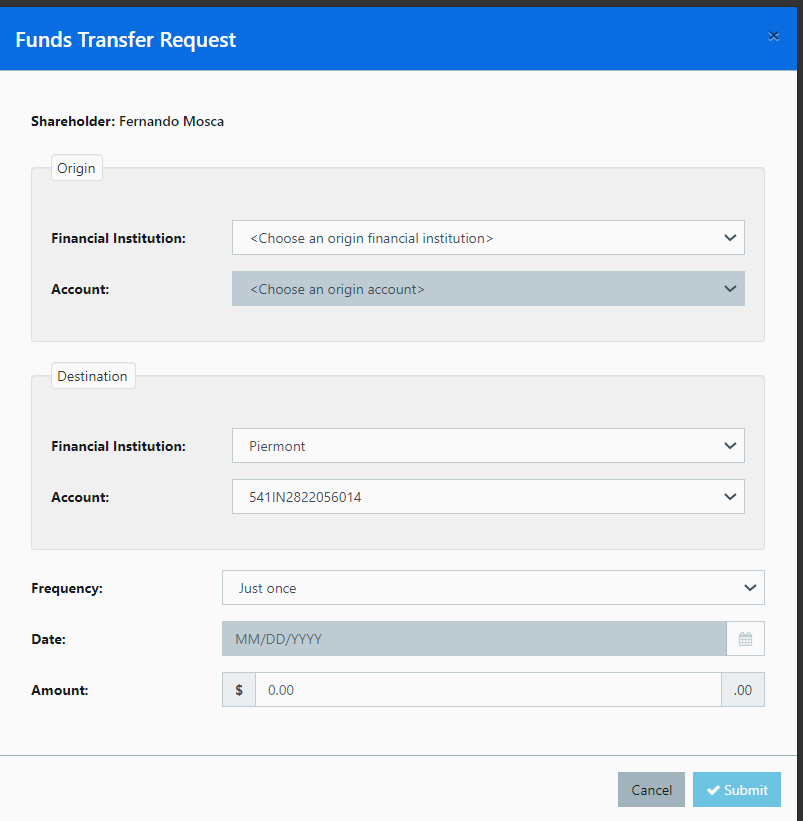
**I.E – Moving funds**

The Request

Being able to fund and account to start trading is maybe the first task that has to be implemented by a new shareholder.

Once logged in, the user will be able to click in the following link:

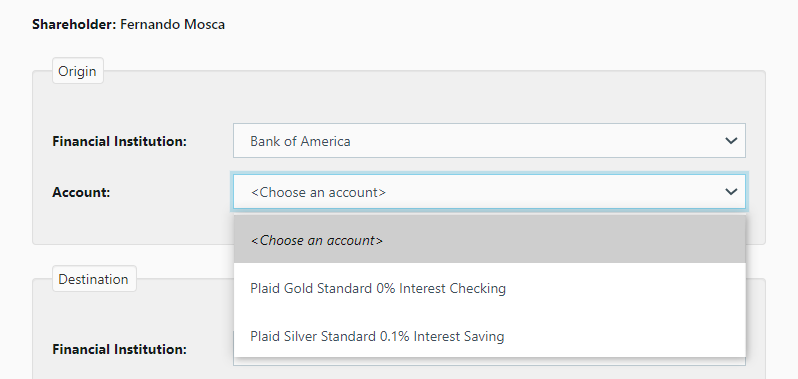
**<Provide link image when the transfer fund request is moved to the new UI>**

****

The images are quite self-explanatory, but we can basically see that there should be a:

* ***Origin Bank Account:*** We will have here all the bank accounts populated after the *Plaid authentication* (see Managing Fund Accounts , #3)

These are the ***Plaid accounts***



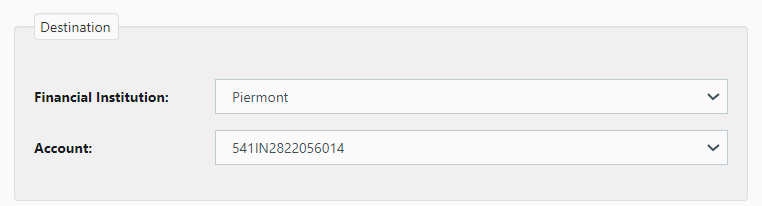
If the user did not provide any external bank account during the onboarding process, this combo will be empty and no transfer will be made.

**In future versions, Plaid account populations will be available in the shareholder’s UI.**

* **Destination Bank Account:**

These will be all the Piermont accounts created in the onboarding process (see Managing Fund Accounts , #2)

If no Piermont account was created in the onboarding process, or if it was rejected, this combo will be empty and no possible transfer will be made.



* **Frequency and amount**

Finally, we have to provide a frequency for the fund transfer request (default is “only once” which is actually the current transfer) and the amount.

At the time of writing this document, only the Only Once frequency allowed.

The amount has to be a positive number and it will be with the real funds available in the real funds available in the Plaid (external) account selected.

Once the submit button is clicked, a confirmation message will be shown and the user will be redirected to the previous screen.

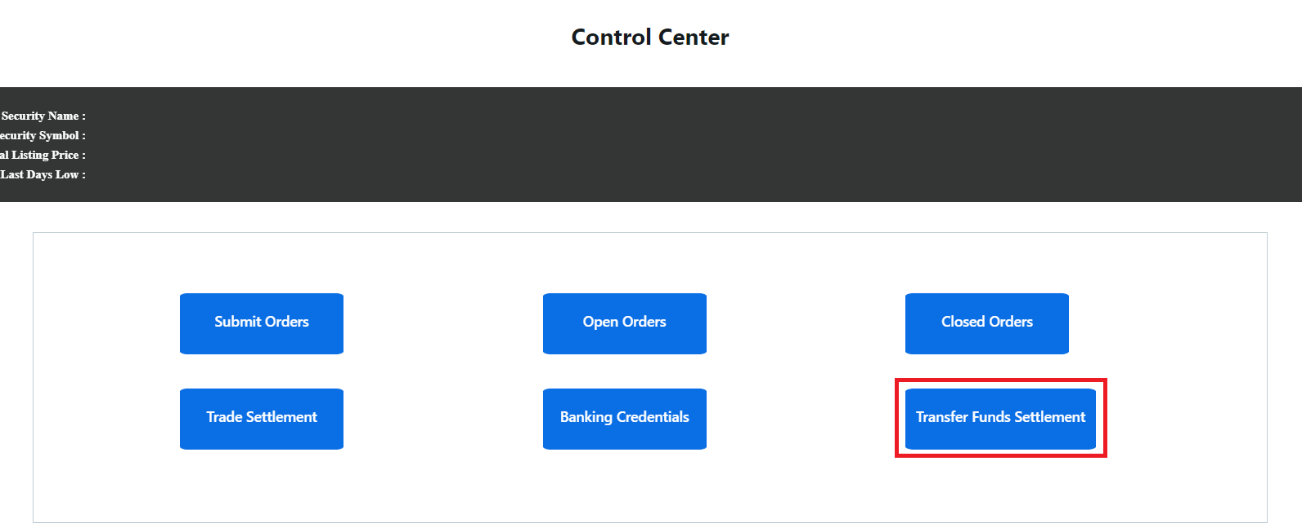
All these transfer fund requests will be stored in table ***transfer\_fund\_requests***.

The Settlement

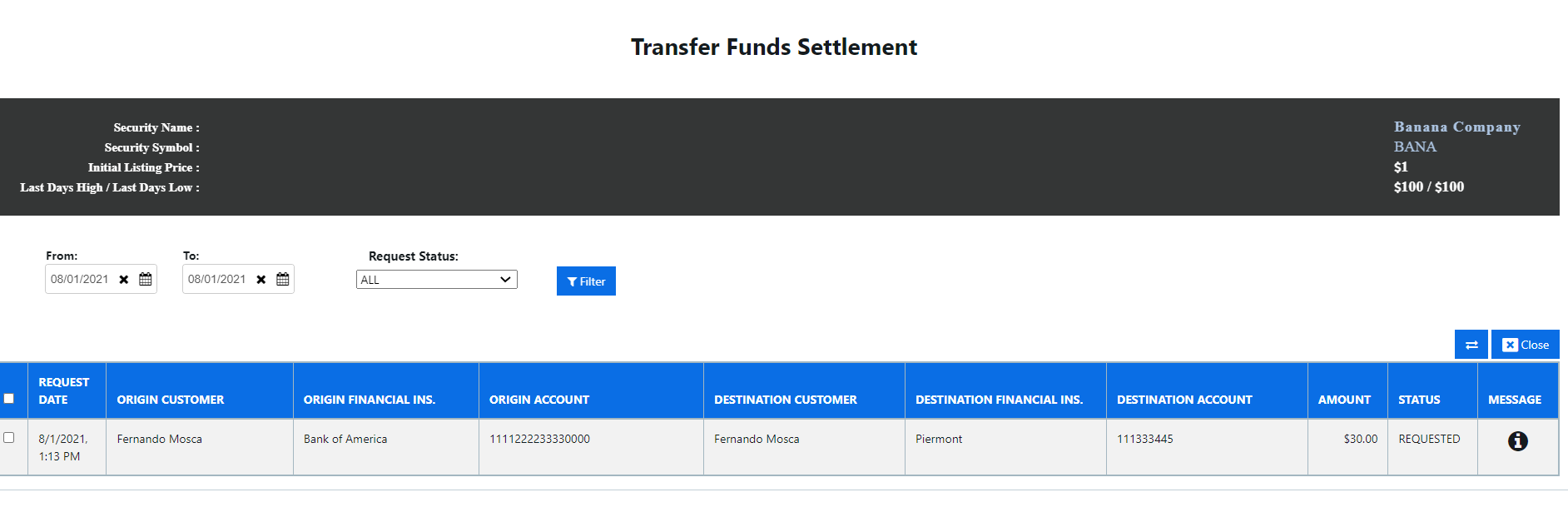
The previous transfer fund request is, as was mentioned, just a request. This means, no NACHA file o transfer fund request was actually executed after clicking the Submit button.

In order to clear this requests, so that the NACHA files are actually sent, in the administrator UI, a Rialto operator has to go to the following screen in the ***Control Center***.

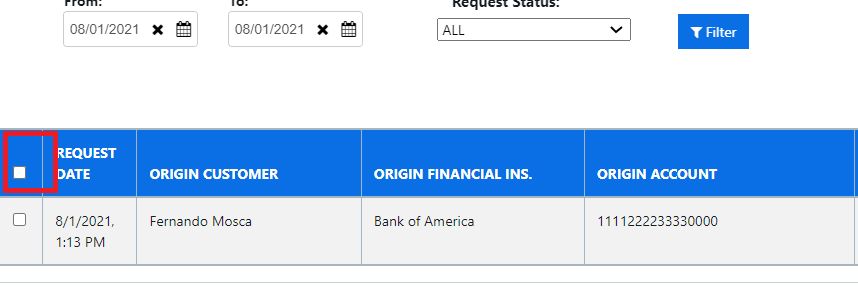




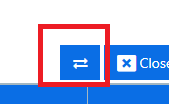
Here we can see the entire current day transfer fund requests, although we can filter the requests by specific date or status.



The settlement can be done, request by request, or all the available requests clicking the following checkbox:

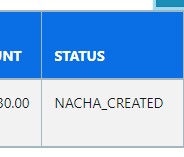


Once all the requests are selected, you just have to click the following button

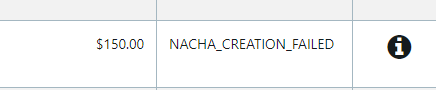


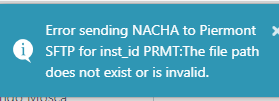
Once that button is clicked, the following steps will take place:

* A NACHA file with all the transfer fund requests will be created
* It will be sent to the Piermont SFTP server
* The transfer fund request will be marked as NACHA created. The funds will be available in the new account and synchronized to the Rialto ATS platform by the time the next *Balance Synchronization* process takes place.



If something fails, you can click the following button and the error that took place will be shown:





**At the time of writing this document, there is not a process to troubleshoot a failed settlement, but new features will be coming in this direction.**

**I.F – Primary Market**

Someone might say that primary market and secondary are both part of the same ecosystem. That is true and it is precisely the primary market the entry point for companies and investors to the platform.

So basically, the natural path to follow for any investment process is

* A new primary company arrives (is onboarded) to the platform.
* A new investor goes to the primary company website and tries to invest in that company clicking the ***invest now button*** present in the company website.
* The payment is recorded and a new KYC process is triggered
* When the KYC process is approved, a Rialto operator has to go through a 4 steps process that will allow the system to fully implement the investment process
  + Trigger the payment
  + Transfer the shares or create a cap table report to communicate to the transfer agent

The previous process is pretty simple and quite similar to any purchase, with the main difference that there will be several actors involved

* ***Jumio*** : Company specialized in KYC processes responsible of validating the users personal information and documentation
* ***Plaid***: Company in charge of managing the whole banking interactions (balance validations, transfers, etc.)
* ***Stripe***: Company that manages different kinds of payment flows. In the context of the primary market, it will be the credit card payments the ones that will be handled through this vendor.
* ***The transfer agent***: At the moment of writing these lines, the communication with the transfer agent is done through an export file**,** but the intention is to automate this interaction in future versions.

Therefore, the primary market functionalities, even when they are smaller than the secondary ones, are very intensive interacting with other vendors and platforms and require a lot of technical knowledge to keep them properly deployed.

So, following we will describe the most important functional and technical aspects of this workflow, following a functional path but not skipping all the technical details necessary to keep all the functionalities properly functioning.

**I.F.i – Primary Company onboarding**

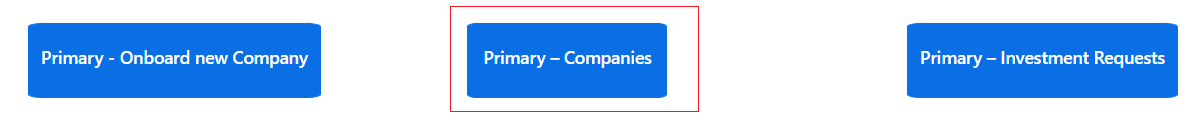
This functionality allows to onboard a new primary company and automatically have ready to be implemented, the Invest Now popup which is the entry point of all the investments.

Here we will populate

* The company name, symbol and logo
* Upload the subscription agreement
* The company main administrator
* The company regulation

After all the previous is done, the company will accessible in the ***Primary Companies*** section, and

* Trigger the payment process from the Primary companies admin section
* Deploy the invest now panel in any Wordpress site available

****

**I.F.ii – Primary Investment Requests – Invest Now popup**

The main goal of the whole investment process is to allow different shareholders to enter their personal information, they payment credentials so that layer the payment and shares transfer can be implemented.

This is achieved through a generic popup where the user goes through different screens, providing different pieces of information until the final step is reached where all this information is submitted for persistence and validation.

Following we describe the existing screens

Screen #1

Here the user will load

* Personal contact information like email and mobile number
* Amount or number of shares to invest

Screen #2

Here the user will provide

* Personal information like name, address and national id
* Documentation numbers like driver’s license or passport

Screen #3

This screens will be mostly dedicated to the payment methods

* **<TO COMPLETE after the latest designs have taken place>**

Screen #4

Here the investor will

* Provide documentation to proof his identity (passports, documents, etc.)
* Take a picture to validate that he is the one in the documents provided

All this will be later used to run the proper KYC procedure by *Jumio* and also by the Rialto operators.

Screen #5

Here the investor will

* See the different subscription agreements and broker dealer agreements
* Validate that he approves those agreements checking a checkbox
* Enter his signature

Payment Rails – Investment Request Service

After all the previous steps have taken place and the *Submit* button has been pressed in the ***Screen #5*** of the ***Invest now*** panel, next step is to run the proper KYC process, which will allow the system to continue with the settlement process.

This will be done in 2 steps

* Submit the ***investment request***
  + After the user clicks the submit button, all the loaded information will be persisted in the database.
  + This includes
    - ***primary\_investment\_request*** table with all the investment parameters like amount and number of shares
    - ***primary\_investor*** table where all personal data loaded during the investment process is saved.
    - ***primary\_id\_verification*** table where all the information extracted from the investor documentation might be persisted. However, at the time of writing these lines, no other information than the one typed by the investor will be persisted in the database.
    - **primary\_stripe\_credentials** table where all the shareholder ***Stripe*** tokens and encrypted and persisted. These will be used to trigger the payment later in the investment process.
    - **primary\_plaid\_credentials** table where all the shareholder ***Plaid*** tokens and encrypted and persisted. These will be used to trigger the payment later in the investment process.
* Run the KYC process
  + This is done by **Jumio** and the investment request status can be fetched In their ***Net Verify*** service
  + To accomplish this, there is a net core job called ***JumioIDVerificationApp***
  + This app will fetch all the investment request that have been sent to ***Jumio*** (***KYC = CREATED*** status), fetch their statuses invoking the Jumio services
    - For this, it will provide the ***Jumio Scan Reference*** token acquired when creating the *investment request* in the ***Invest Now*** panel.
    - If the investment request has been successfully approved by Jumio, it will update its status to ***APPROVED***
    - If it had been rejected for some reason, the status would have been ***REJECTED***
  + So basically , after this whole process hast taken place, we will have investment requests that have been rejected or approved,

**I.F.iii – Primary Investment Request - Settlement Dashboard**

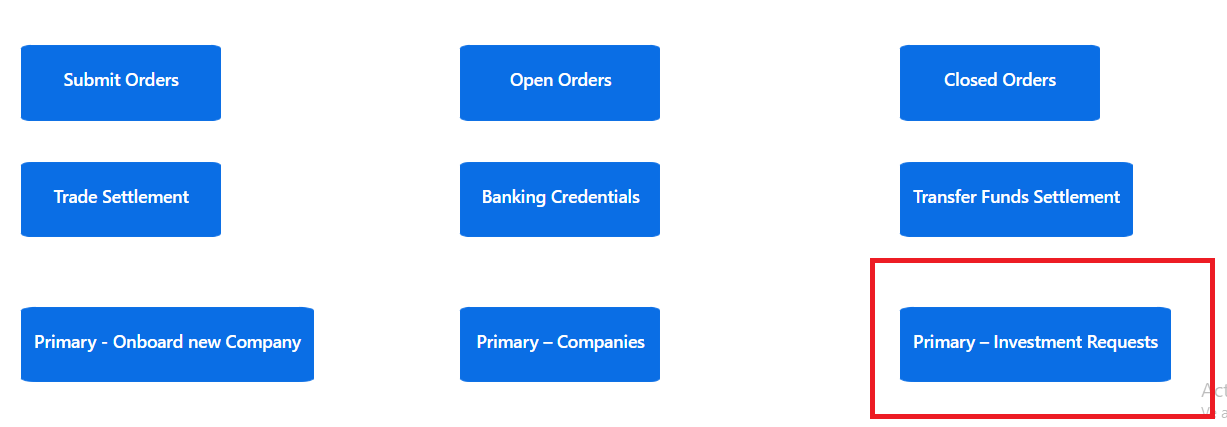
Once all the investment requests have been received, we need a process to allow a ***Rialto*** operator to clear all those investments. The process will have to

* KYC all the existing shareholders rejecting or approving the investment based on the best *Know Your Customer* practices.
* Move all the funds from the investor account, credit card, etc. to the company escrow account.
* Create a ***cap. table*** report to be provided to the transfer agent, so as the shares can be properly deposited in the investor’s account
* Finally mark the investment as settled.

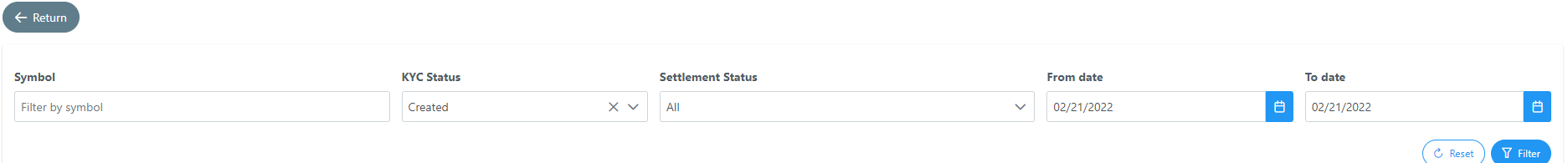
Following we will describe all the previous steps and the screens that

When a new investment request is received, a Jumio ID Verification process will check with Jumio if the investment has been approved or rejected (see ***II.E.v - Jumio ID Verification)***

So if we go to the following screen



We will see the following filters

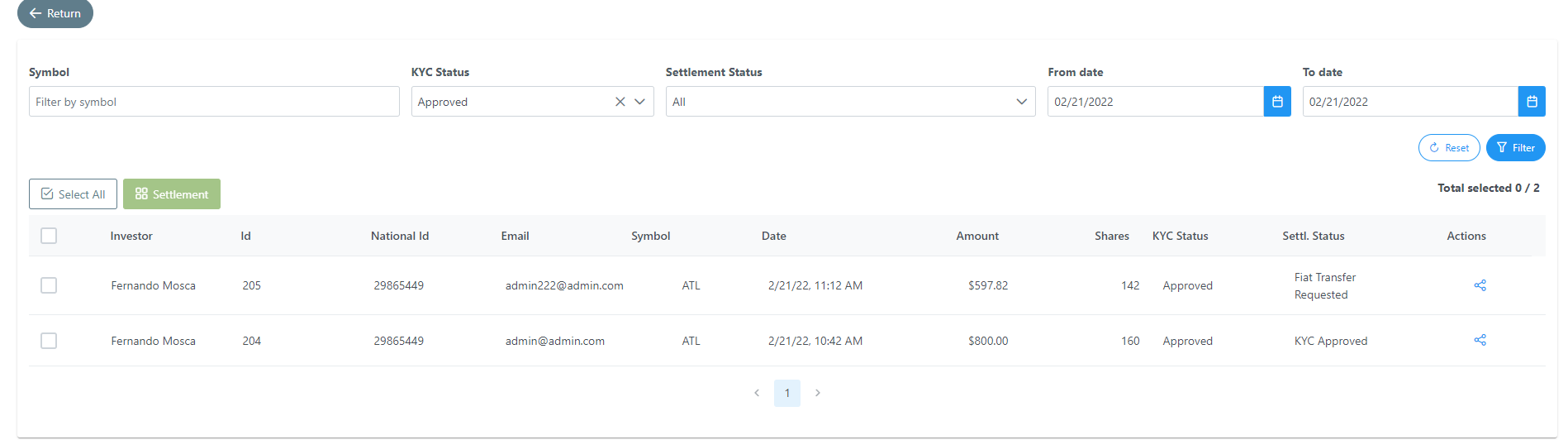


Where

* **Symbol**: Filters the symbol of the primary market security to settle
* **KYC Status**: KYC Status in relationship to the Jumio identity validation. There can be 3 statuses
  + Approved: The application has been approved and it is ready to start. **Only the applications marked as APPROVED status will be able to continue the Settlement process** (Settlement button enabled)
  + Rejected: Jumio has rejected the application. Further information about this will have to be retrieved from the Jumio portal.
  + Created: Jumio is still validating the new shareholder credentials
* **Settlement Status**
  + Settlement status once the application has been approved. GIven that an investment settlement requires transferring the fiat money and then transferring the shares, this combo has 4 specific steps.
    - **KYC Approved:** The KYC has been approved and It is ready to transfer the money (Stripe, Plaid, etc.)
    - **FIAT Transfer Requested:** FIAT money transfer has been requested and received by the payment manager but not yet confirmed
    - **FIAT Transfer Confirmed:** FIAT payment has been confirmed by the payment manager (Plaid, Stripe, etc.)
    - **Shares Transfer Requested:** Cap. Table export file has been created
    - **Settled**: The trade has been settled
* **From/To**: Date filters for the primary investment request date

So if we want to start clearing a given number of investment request, we will have to pick the ***APPROVED*** option n the ***KYC Status*** combo and then start navigating through the different **Settlement status**.

Lets make this clearer with an example. A typical Rialto operator journey might start going to the Settlement Dashboard and requesting all the investment requests that have been approved for today



Of course, as you ca see, the ***Settlement button*** is disabled, because we are filtering ***All*** the ***Settlement status***. So we will have to pick a specific settlement status if we want to trigger a specific action.

Requesting fiat to be transferred

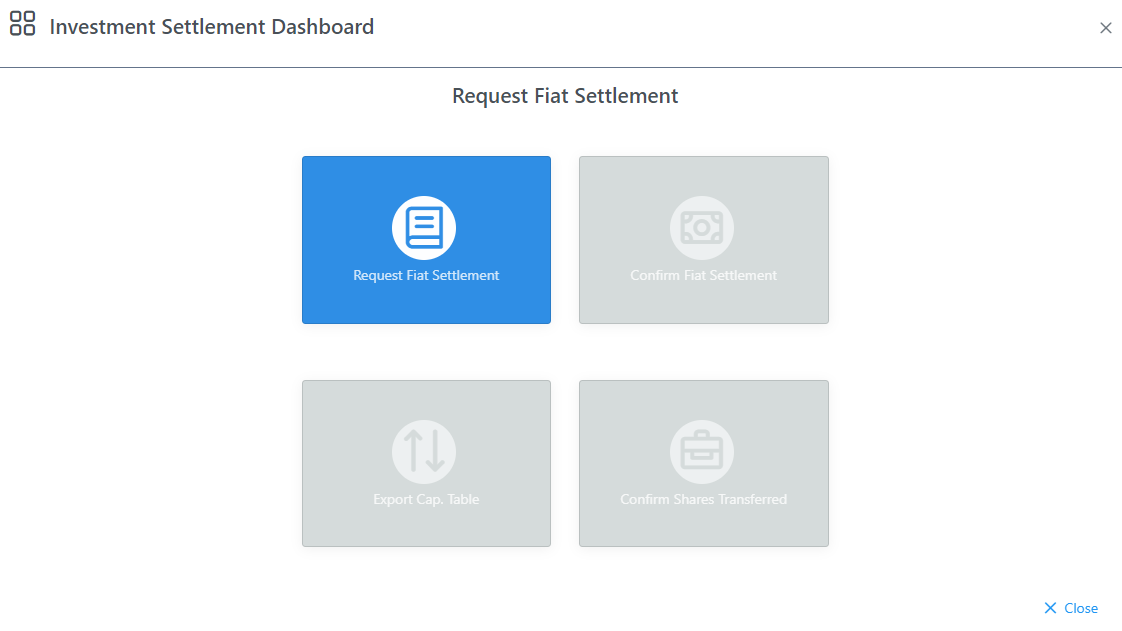
So if we want to request the fiat to be transferred we will pick the following filters for the ***Status*** combos



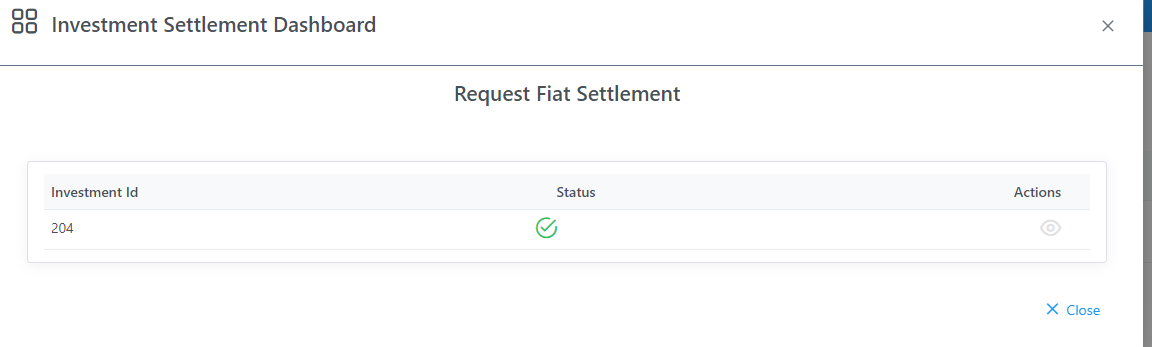
Then we have 2 options (and this applies to the four settlement steps)

* We can pick ***Select All*** and all the investment requests in the current status **for the selected date range** will be cleared when picking the Settlement button.
* We can pick the investment requests we want to settle one by one, and then click the ***Settlement*** button.

Either way, we will see the following screen when clicking the Settlement button



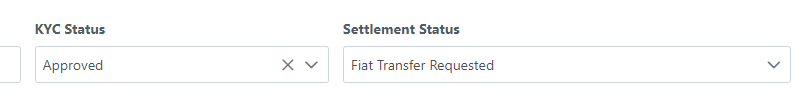
After clicking the blue button, all the selected investment requests will be picked and the *Payment Rails* will communicate with ***Plaid , Stripe*** or whatever payment manager that is linked to every investment request and request the fiat transfer to be executed. This can succeed or fail and such response will be shown in the following summary-



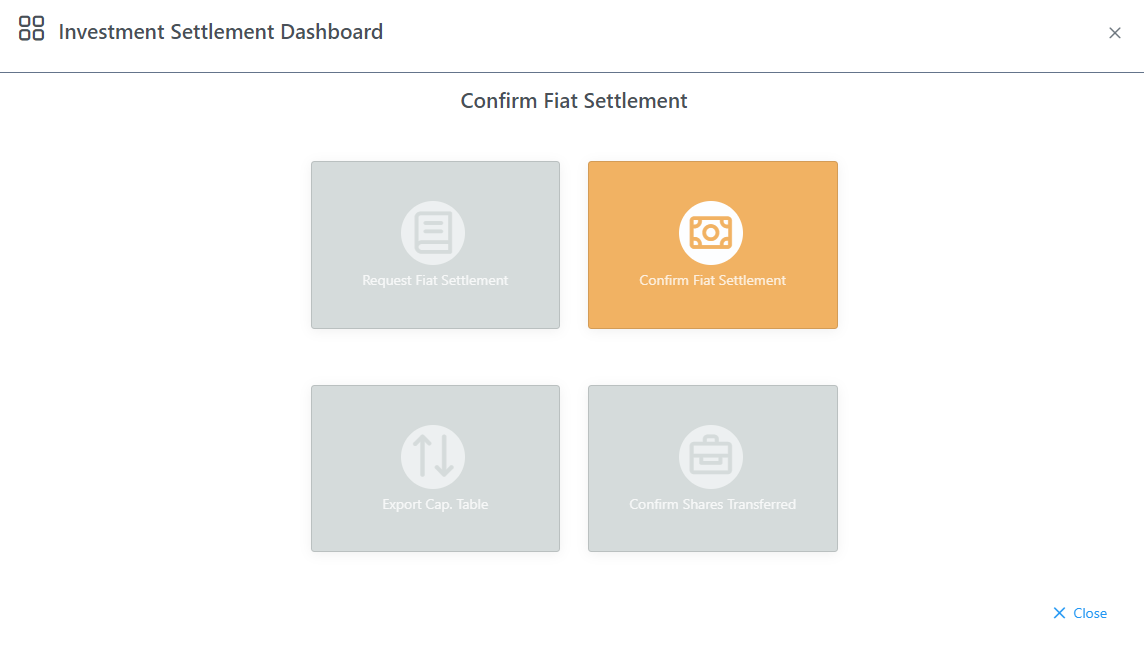
Note: If there had been an error, the ***Status*** column would show an error link and the ***Actions*** column would show a popup describing the error.

Confirming that fiat has been transferred

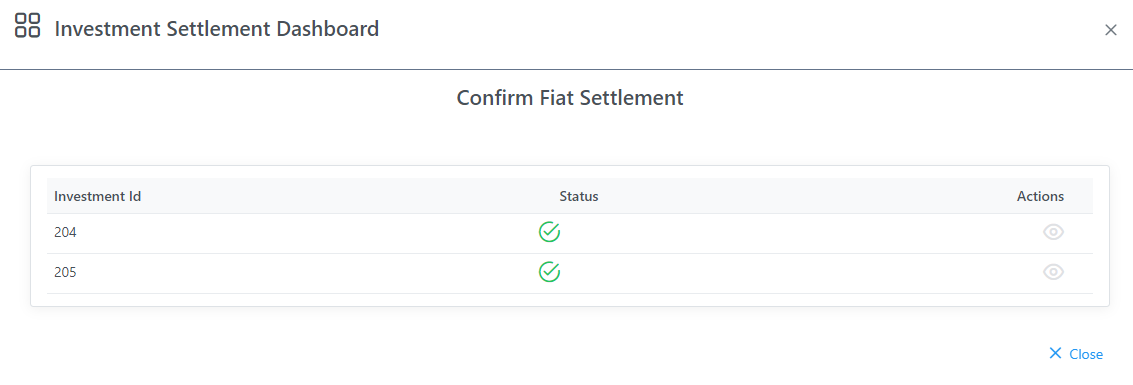
If we want to confirm that the fiat has been transferred we will pick the following filters for the ***Status*** combos



When clicking the ***Settlement*** button (by date range or specific investment requests) we will see the following screen when clicking the ***Settlement*** button



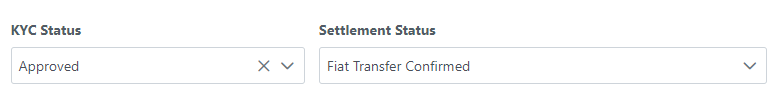
After clicking the yellow button, all the selected investment requests will be picked and the *Payment Rails* will communicate with ***Plaid , Stripe*** or whatever payment manager that is linked to every investment request and confirm that the payment has been triggered. This can succeed or fail and such response will be shown in the following summary-



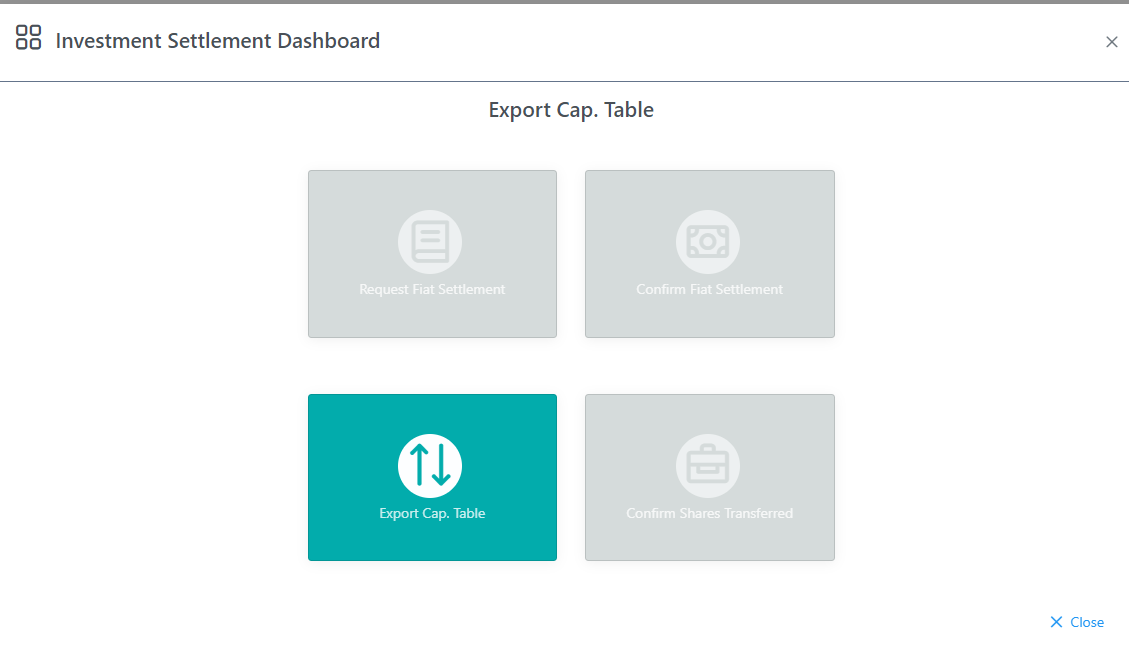
Note: If there had been an error, the ***Status*** column would show an error link and the ***Actions*** column would show a popup describing the error.

Cap. Table file creation

If we want to create the cap. Table file to be sent to the transfer agent, we will pick the following filters for the ***Status*** combos



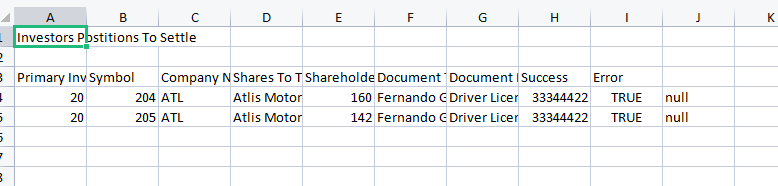
When clicking the ***Settlement*** button (by date range or specific investment requests) we will see the following screen when clicking the ***Settlement*** button



After clicking the green button, all the selected investment requests will be picked and the *Payment Rails* will issue a cap. Table report that will have all the shares that have to be transferred to every specific investor. A cap. table file will be recorded in the database, so these cap. table files cannot be created twice avoiding a double shares transfer record. **Future functionalities will allow to re export these cap. tables.**



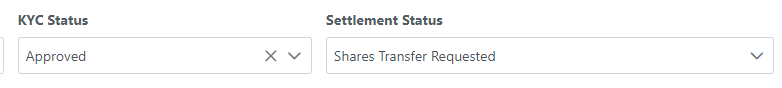
If we click the ***Download CSV*** button, we will download the CSV file.



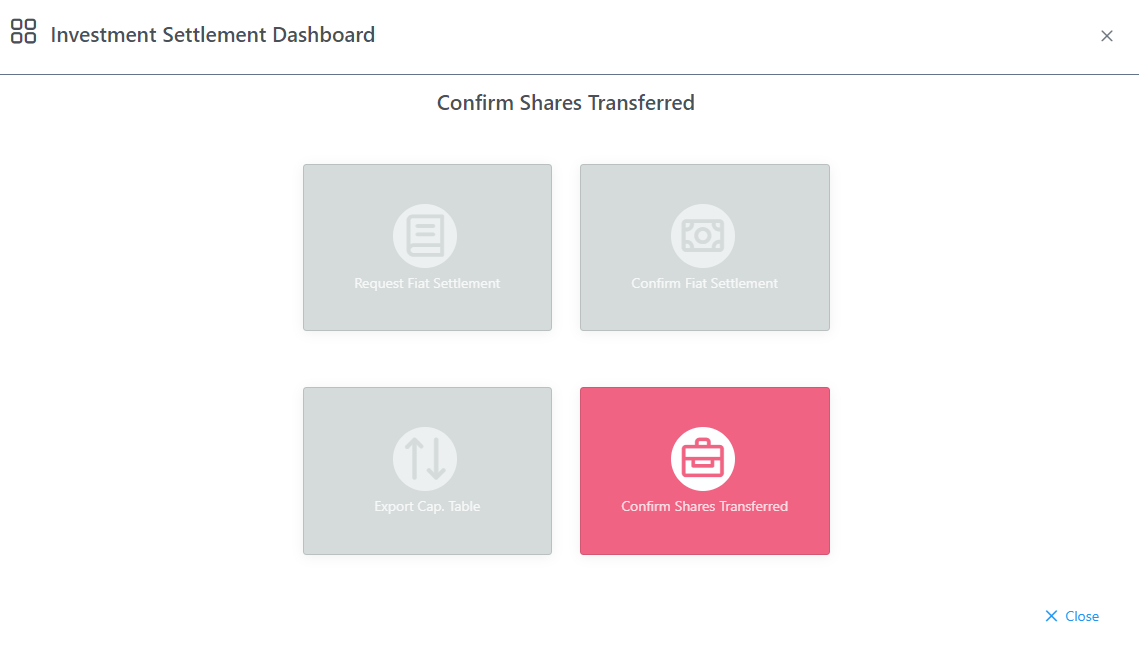
Confirm that the shares have been transferred

This is just a formal step in case that we have some day a transfer agent that has an asynchronous shares transfer process. But so far, as the shares transfer is manual this step will only allow to mark every investment request as ***Settled***.

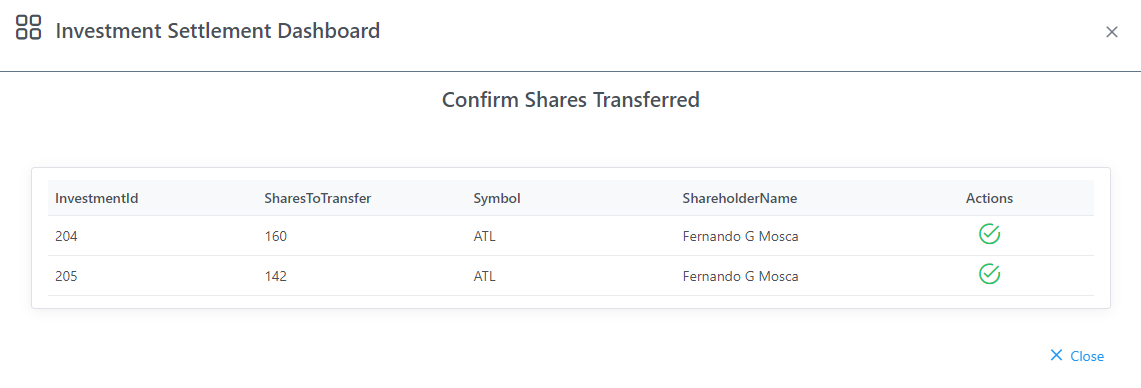
To accomplish this, we will pick the following filters for the ***Status*** combos.



When clicking the ***Settlement*** button (by date range or specific investment requests) we will see the following screen when clicking the ***Settlement*** button



After clicking the pink button, all the selected investment requests will be picked and the *Payment Rails* will mark the investment request as **Settled**.



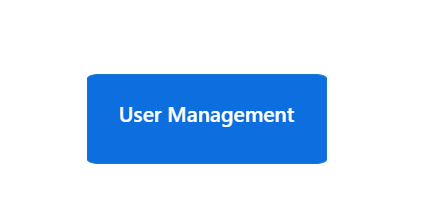
**II – Technical artifacts**

**II.A - Users**

As was previously mentioned, the ATS has 2 different levels of users

Administrator user: These users have the power to manage all the main backend functionalities that exist: create securities, other users, clear trades, clear transfer funds requests, etc.

They are created in the following tab





Shareholder user: These users are redirected to a whole different UI when logged in as they are the users to who the platform is dedicated to. Basically they can trade and have access to all the securities main links and graphs. They can be onboarded manually, but the most common scenario is to do it through the transfer agent (Ex: ***Kore Con X***) or through ***Solidus*** in the onboarding workflow (see document ***Rialto KCX Solidus integration - Functional specs v1.x***as an example of all the steps of all this onboarding process implements)

**II.B - UI**

<TO COMPLETE>

**II.C- Backend**

<TO COMPLETE>

Sensitive data – Logging

As was mentioned before and described throughout the document the Rialto ATS will be interacting with several applications, own or third party, and this means that a lot of action will be taken under curtains which makes so important having a mechanism to track and troubleshoot all these interactions.

In this context, at the time of writing these lines, there were not a UI dedicated to tracking some sort of logging mechanism but the tables and endpoints existed.

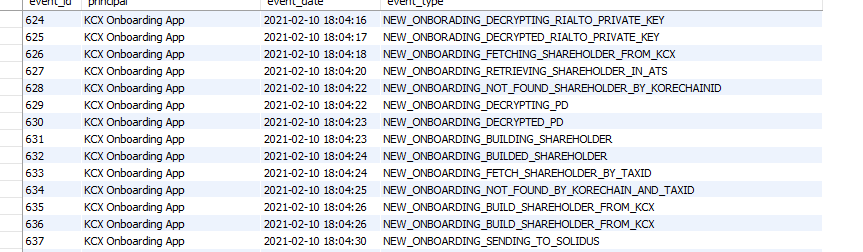
These tables are:

* + ***jhi\_persistent\_audit\_event***
  + ***jhi\_persistent\_audit\_evt\_data***

The mechanics of both tables is pretty simple, and reading ***jhi\_persistent\_audit\_event*** rows allows us to know different actions that took place throughout the execution of a given functionality.

It´s worth to mention that any logging in an application is very dynamic and everything that might be written one day will soon become outdated. So we will give a brief description about these tables through some example. But keep in mind that the main concepts described in this example, should apply for any other functionality that uses these tables.

The first example belongs, precisely to the onboarding service. Taking a look to the **jhi*\_persistent\_audit\_event*** tablewe can see that for an onboarding process, several rows, will be added with a title that can pretty much give an idea about what was the action referred in that row.



But then we can pick one specific row, and go into further detail using table ***jhi\_persistent\_audit\_evt\_data.*** If we take for example event 628 we can see that every event will be describe through 3 basics fields



* **id\_name** and **id\_value**: Every logging action will be related to something. This can be a security, a service, a shareholder, etc. Sometimes this shareholder could be identified by its firm id, his email or tax id. Other times, like the situation when someone is being onboarding and not much is known about him, it will be identified by exotic things like his *kore chain id*. Either way, in every recording, we will try to identify the entity affected by such action. So for every recording, the ***id\_name*** will describe the id that it´s used to identify the referred entity and the ***id\_value*** field, will precisely have such id.
* **Message:** Having identified the entity, we must have something to tell about it and a specific action that was taken in that pint in time. This will be recorded under the ***message*** key. In the previous example we can see that in an onboarding process, the onboarding service is informing that the onboarded shareholder, could not be found by the provided *kore chain id*. So at least, based in the received ***kore chain id,***  it´s a new shareholder.

At the time of writing this lines, there was not a UI that could create reports that would allow to have access to all of these records, but this will be one of the most important features to implement in the near future.

**II.D - Database**

Settings tables

Different settings have to take place to enable the proper implementation of different functionalities. Some of these are mostly related to different business parameters (like Rialto’s Tax Id), and some others are more related to technical features, like the table that specifies all the net core and external services used in one way or the other by the ATS.

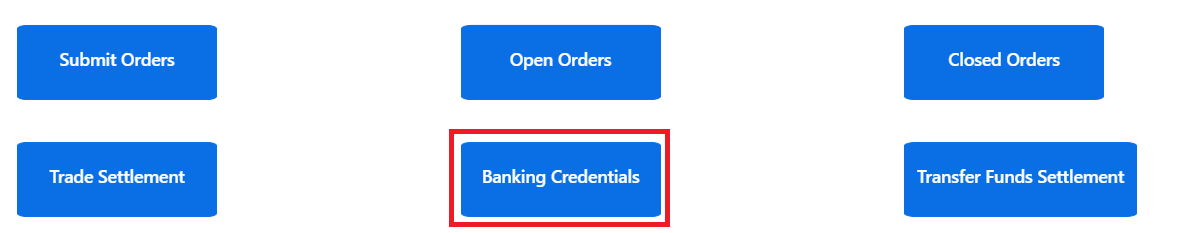
* **system\_variables**
  + It stores several specific business parameters used by the application
  + The most important are
    - RIALTO\_NAME: Rialto Official Name
    - RIALTO\_TAX\_ID: Rialto official Tax Id
    - TRADE\_COMMISSION\_DEFAULT\_PCT: Commissions to be applied when matching trades
    - PIERMONT\_TRANSIT\_ROUTING\_NUMBER: Transit Routing number when creating NACHA files that have deposits in a Piermont account
    - ONBOARDING\_STORE\_INCOMING\_MESSAGES: All the incoming messages to the net core services will be logged in the ONBOARDING\_INPUT\_MESSAGE\_TEMP\_FOLDER folder. This is for testing purposes.
    - CLOB\_MATCH\_REFRESH\_SECURITIES\_FREQ\_MILISEC: Is the frequency (in miliseconds) a newly added security will be incorporated to the matching algorithm buffer
    - CLOB\_MATCH\_RUN\_MATCH\_MILISEC: Is the frequency (in miliseconds), a matching takes places in the order book
    - CLOB\_MATCH\_REFRESH\_ORDERS\_FREQ\_MILISEC: Is the frequency (in miliseconds) an order entered through the ATS will be added to the matching algorithm buffer
    - ACCOUNT\_CONF\_CREATION\_EXEC\_TIMES: CSV (comma) separated times where the Piermont account creation confirmation packages will be downloaded and processed.
    - BALANCE\_SYNC\_EXEC\_TIMES: CSV (comma) separated times where the Piermont account balances will be synchronized.
    - AWS\_ACCESS\_KEY\_ID/ AWS\_SECRET\_ACCESS\_KEY: Amazon SQS queues credentials
    - AWS\_REGION: Amazon SQS queues zone for regional calculations (ex: time zones)
    - CLOB\_WEBSOCKET\_URL: The other way to communicate with the CLOB engine (besides an SQS queue) is a websocket port. This Is the URL that the CLOB will be listening for websocket connections.
    - FIX\_LOGS\_FOLDER: Folder where all the FIX logs of the FIX layer of the CLOB engine, will be persisted.
    - FIX\_SESSION\_FOLDER: Folder where the *acceptor.cfg* file of the CLOB Engine can be found.

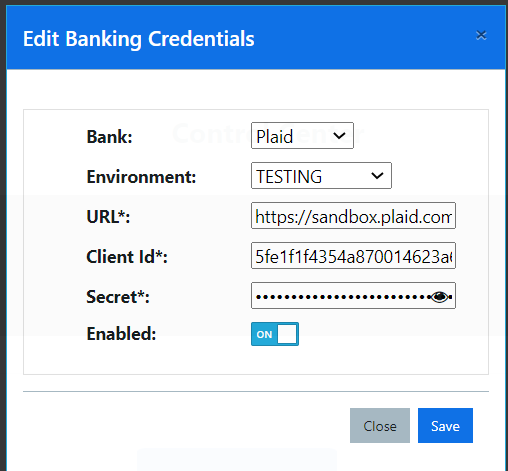
***Note: It is expected to encrypt all these values in the near future****.*

* **external\_servers**
  + It has a list of all the external servers used by the platform. At the time of writing this document we have the following servers
    - SFTP\_ACCOUNT\_STATUS\_PIERMONT: Used to retrieve the account creation confirmation file. This is the file that will be used to confirm that an account has been created
    - SFTP\_NACHA\_PIERMONT: This is the server that will store all the NACHA files sent to Piermont
    - AWS\_CLOB\_SQS: Amazon SQS Queue used to communicate the ATS with the CLOB engine (subscribe to the order book, subscribe to a specific order, create orders, cancel orders, etc.)
  + The most important fields are
    - Environment
      * DEV/PROD
      * The net core services will be executed in a test or productive mode. <Complete this when the file/environment variables are implemented>.
      * So basically we should use this field to indicate that the current record is a real or productive SFTP server
    - address: IP address or URL of the SFTP server
    - port: Port of the SFTP server
    - user/password: credentials to log in to that server. It is expected to encrypt these values by the time the AWS environment variables are implemented.
    - default\_folder: It indicates if the net core service should change directory to the current selected folder, when interacting with this specific server.
* **external\_services**
  + This table stored the name and ***url/ip*** addresses of all the net core services used by the ATS platform.
  + It will be the service the net core service that defines if the service is used in a test or productive environment (see **external\_servers**)
  + The most important services at the time of writing this document are
    - BANKING: It has all the Plaid and NACHA file created services used by the ATS platform.
    - POSITIONS\_SYNCHRONIZATION\_SERVICE: Used by the ATS to synchronize a shareholder’s positions with the different transfer agents when logging in to the platform.
* **solidus\_settings:**
  + This table has the solidus onboarding services URLs that are invoked in the onboarding process
  + It also has record that refer to test environments (***DEV***) or productive environments (***PROD***).
  + Basically this table will have
    - login\_url: The url used to authenticate towards the **Solidus** environment
    - opx\_url: The url used to send the different onboarding requests
    - login/password: These are the Solidus credentials and will have to be encrypted in the near future.
* **plaid\_settings:**
  + This table has the Rialto credentials in the Plaid platform that will be used in every Plaid services invocation.
  + These credentials imply a *client\_id* and a *secret*.
  + The *env\_name* can have the following attributes
    - SANDBOX
    - TESTING
    - PROD

The first two are test environments , but with the only differences that you can only use real bank accounts in the TESTING environments, and fake access tokens in the SANDBOX environments

* + The *url* can have the following attributes
    - [*https://sandbox.plaid.com*](https://sandbox.plaid.com)
    - [*https://development.plaid.com*](https://development.plaid.com)
  + These values are not encrypted at the time of writing this document and this should be done in the upcoming sprints.
  + These values can be managed through the following screens

**

**

* **jumio\_settings:**
  + This table has the Rialto credentials and of the *Jumio* platform that will be used for running *KYC* processes.
  + These credentials imply a Rialto’s *token* and a *secret*.
  + There are 2 urls that will be used
    - *auth\_url:* 
      * Used to authenticate Rialto with Jumio
      * [*https://auth.amer-1.jumio.ai/oauth2/token*](https://auth.amer-1.jumio.ai/oauth2/token)
    - *opx\_url:*
      * Used to run the download the KYC statuses
      * https://netverify.com
  + The *env\_name* can have the following attributes
    - DEV
    - PROD
  + In the different environments , the credentials should be the same
  + These values are not encrypted at the time of writing this document and this should be done in the upcoming sprints.
* **stripe\_settings:**
  + This table has the Rialto credentials and of the *Stripe* platform that will be used for managing credit cards payments for the Primary Market.
  + These credentials imply a Rialto’s *customer\_token*.
  + There is 1 url that will be used
    - *url: https://api.stripe.com*
  + The *env\_name* can have the following attributes
    - DEV
    - PROD
  + These values are not encrypted at the time of writing this document and this should be done in the upcoming sprints.
* **institutional\_accounts:**
  + This table has all the Rialto institutional accounts that will be used throughout the life time of the platform. For example, the only institutional account that exists at the time of writing this documents is the following:
    - SETTLEMENT\_CONTROL\_ACCOUNT: Used to deposit all the commissions charged when opening an account and clearing the trades
* **kcx\_connection\_settings:**
  + This table has all the credentials that belong to ***Rialto*** in the sense of its relationship with the ***KoreConX*** transfer agent. This means
    - ATS\_ID: Identifier of the ***Rialto*** ATS in the *Hyperledger* platform. It will be requested by many of the Hyperledger REST services when running a POST or GET request
    - URL: URL of the Hyperledger node that the ATS should interact with
    - User/password: User and password that should be used to authenticate with the REST services in the Hyperledger.

**II.E- Net core services**

It was mentioned that the main backend technology is Java, but this does not mean that it is the only technology in the stack. For its diversification advantages and the high level of portability in its newest version, many functionalities of the ATS were built in .net core.

The criteria that was used to build a specific functionality in this technology was business centric (see ***IV- Development Process***). This means, if the functionality involved many interactions with external servers, a broad range of external servers or high frequency functionality, then net core was the technology chosen.

At the time of writing this document, the following are the services/ applications that had been built using this functionality

**II.E.i- Onboarding Jumio ID Verification**

OnboardingApp

This is a service that will be opened by default in ports 27900, 28900, 29900 (dev, staging, uat, etc.) by default. It is responsible of managing all the onboarding requests started in the different transfer agents. At the time of writing this specs, the only available transfer agent was KoreConX.

All the steps described in section “**3-Onboarding”** of document ***Rialto KCX Solidus integration - Functional specs v1.x*** will take place here, so refer to that document for more details.

PlaidCredentialsLoadApp

This service is used to provide the ***Plaid*** tokens for started or non-started applications.

It will be opened by default in ports 27901, 28901, 29901 (dev, staging, uat, etc.) by default.

At the time of writing this document, the only existing transfer agent is ***Kore Con X,*** so for more information refer to the section “**3-Onboarding”,** Solidus - Plaid Onboarding of document ***Rialto KCX Solidus integration - Functional specs v1.x***

ApplicationApprovalService

This service is used to approve previously existing onboarding applications (already started applications) or to start a new one completely from scratch.

It will be opened by default in ports 27902, 28902, 29902 (dev, staging, uat, etc.) by default.

As was mentioned, when this service is used to approve existing application, as at the time of writing this document, the only existing transfer agent is ***Kore Con X,*** you can have a better understanding of what takes place reading section “**3-Onboarding”,** Solidus - Application Approval of document ***Rialto KCX Solidus integration - Functional specs v1.x***

However, as was mentioned before, this service is also used to start application for shareholders whose application was triggered from ***Solidus*** and not from ***KoreConX***. When this situation takes place, this service will check that there is not an open application for the incoming email, and it will take the following actions:

1. Create an onboarding application for the incoming email
2. Create the newly created shareholder with his email and password created at Solidus as login credentials
3. Approve the application created in step #1
4. If this new shareholder Plaid credentials had arrived previously, it will map the previously received credentials to the new shareholder created in step #2
5. Create the account creation package and send it to ***Piermont***

From this point on , the shareholder will be able to log in into the platform but not to fund his account until his new ***Piermont*** account is approved and he will not be able to trade as well. All his transfer agent credentials will have to be populated manually in this case.

PiermontAccountCreationApp

This service is used to confirm that the ***Piermont*** account requested in step #5 of the application approval process, has been approved by Piermont and it’s ready to be used.

This app will connect to the **Piermont** SFTP server (DEV or PROD environment) based on the specified setting in the ***external\_server*** variables. It will be the ***app.config*** ***TestEnv*** setting the one that will define if a productive or testing server will be used.

**II.E.ii- Banking Services**

The ATS will basically have two specific points where it will need to consume banking services and interact with the different banks.

* When funding a shareholder’s account
* When clearing trades, as it will have to move funds to the sellers and shares to the buyers.

In this context, these services take the responsibility of communicating with the main ATS bank (***Piermont***) and also retrieve different relevant data from the external banks using the ***Plaid*** credentials.

Bank Synchronization

These are the endpoints provided by this service:

* **plaid/user-plaid-info**: Used by ***Solidus*** to provide the shareholder’s external banks credentials populated in the onboarding process. If there is not an existing shareholder when invoking, these credentials will be stored to be assigned when the shareholder is created.
* **Plaid/GetFinancialInstitutions**: Used by the transfer fund request screen to retrieve from ***Plaid*** all the financial institutions linked to the shareholder ***Plaid’s*** tokens.
* **Plaid/GetAccounts**: Used by the transfer fund request screen to retrieve from ***Plaid*** all the accounts associated with a given external bank account linked to the shareholder.
* **Plaid/TransferFundsRequest:** Used by the transfer fund request screen to request a transfer fund which will be executed later in the transfer fund settlement screen by a ***Rialto*** operator.
* **Banking/CreateNACHAForTransferFundsRequests:** Used by the transfer fund settlement screen to request that a NACHA file is created and sent to the Piermont (Fiserv) SFTP server where it will be processed and the ***Piermont*** account will be funded.
* **Banking/CreateNACHAForTradeSettlement:** Used by the trade settlement screen to create a NACHA file that transfers money from the buyer to the seller’s ***Piermont*** account. All the commissions transfer between the buyers and sellers accounts to the Rialto institutional account (see table *institutional\_accounts*, key *SETTLEMENT\_CONTROL\_ACCOUNT*). The commission percentage used will be the key *TRADE\_COMMISSION\_DEFAULT\_PCT* in the table *system\_variables*.

**II.E.iii- CLOB engine**

For a complete description of all the CLOB engine technical and functional features, please refer to the ***FIX + Websocket Layer API - Users manual v1.x.pdf*** document.

**II.E.iv- Payment Rails**

The payment rails is a complex platform that ties to manage in a single point all the different payments that take place when clearing the different investments in the primary market environment.

It is implemented as a set of services in the **Application Approval** service environment (see ***II.E.i- Onboarding***). Each one of this services take care of a specific action when clearing the investment requests.

All these services are consumed form the Primary Investment Request - Settlement Dashboard (see ***I.F. iii - Primary Investment Request - Settlement Dashboard***),so in this section we will mostly focus in describing the service that lye underneath all the functionalities described in that section.

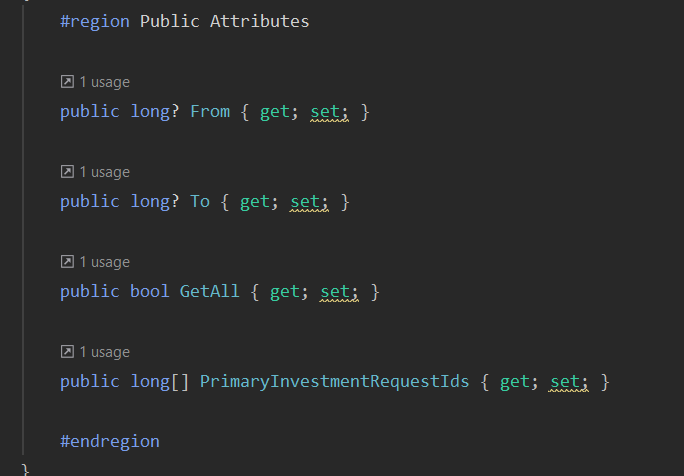
We will also give a brief description of all the interactions with the different vendors and the most important technical considerations that have to be taken into account in each case.

OnFiatLegPaymentRequest service

This is the first step of the settlement process. Basically the system will receive a request to transfer the money for a given number of investment requests and it will have to decide which is the proper channel for which to trigger all those payments (Stripe, Plaid, etc.).

In this context, it is important to remark the following concepts

* Input Json: This is a json that will tell this service to clear a given number or range of investment requests. Basically there are two options
  + We can clear a given number of investment requests. In this case we will have to populate the ***PrimaryInvestmentRequestIds*** array and set the ***GetAll*** Boolean variable to **false**.
  + Another option is to provide a specific range of dates and set the ***GetAll*** Boolean variable to **true**.



* Actions taken:
  + The service will fetch all the existing investment requests that are in the ***APPROVED*** status for the provided filters (see Input Json) at table ***primary\_id\_verification*** related to the ***primary\_investment\_request***)
  + It will fetch all the *settlement* records in the **primary\_investment\_request\_settlements** tablefor the previous ***primary investment requests*** found
  + It will validate that the settlement is in ***KYC\_APPROVED*** status (***status*** field in **primary\_investment\_request\_settlements** table)
  + If it is a ***Plaid*** payment, it will communicate with ***Plaid*** and trigger the bank transfer.
    - **<TO COMPLETEwhen the *Plaid* functionality is enabled>**
    - This is all we will describe in this section. For more details about the ***Plaid*** services refer to section **II.I.ii – Plaid services**
  + If it is a ***Stripe*** payment, it will communicate with ***Stripe*** and trigger a *credit card* payment
    - Different steps will be taken that are necessary in the Stripe environment, like
      * Fetch escrow account for the selected company
      * Get the ***setup intent*** created in the **Invest Now** popup with ***Stripe***
      * Get the ***payment method*** specified in the Invest Now popup with **Stripe**
      * Create a ***payment intent*** with ***Stripe***
      * Update the settlement process (**primary\_investment\_request\_settlements table)** and its detail (***primary\_investment\_request\_settlement\_details table***) with the result of the previous operation.
  + If it detects any other payment method, the fiat settlement will be rejected as there are not any other method at the time of writing this document.
* After all the investments requests affected by the input json are processed, some of them will have been successfully processed and some of them will have found some sort of issue/error. A json will be returned specifying for every investment request, the result of its fiat settlement.



OnConfirmSharesTransferred service

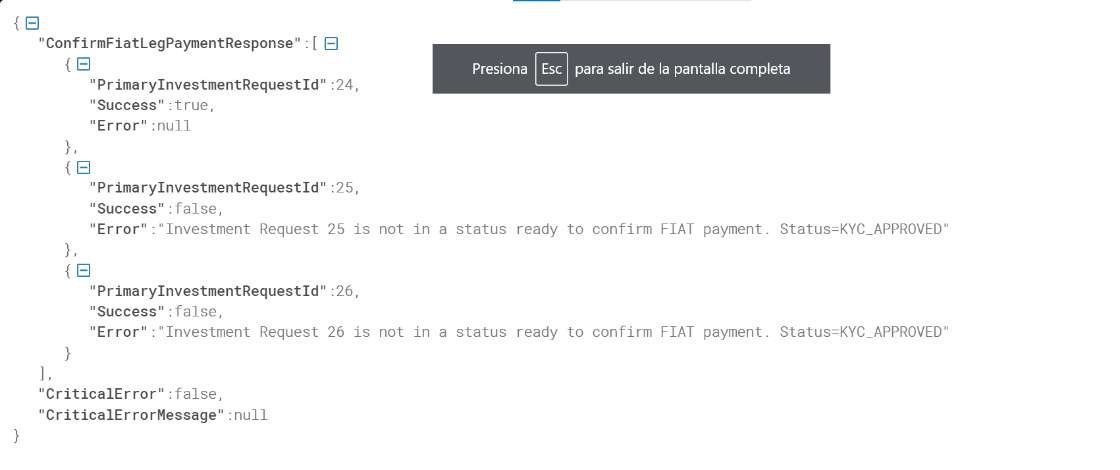
This is the second step of the settlement process. Basically the system will receive a request to confirm that all the money transferred in the previous steps has been successfully transferred. In order to achieve this, the system will have to communicate with all the different vendors (***Stripe, Plaid, etc***.) and confirm the different transfer statuses.

In this context, it is important to remark the following concepts

* Input Json: This is a json that will tell this service to clear a given number or range of investment requests. Refer to the *OnFiatLegPaymentRequest* service to see how all the parameters of the input json work.
* Actions taken:
  + The service will fetch all the existing investment requests that are in the ***APPROVED*** status for the provided filters (see Input Json) at table ***primary\_id\_verification*** related to the ***primary\_investment\_request***)
  + It will fetch all the settlement records in the **primary\_investment\_request\_settlements** tablefor the previous *primary investment requests* found
  + It will validate that the settlement is in ***FIAT\_TRANSFER\_REQUESTED***  status (***status*** field in **primary\_investment\_request\_settlements** table)
  + If it is a ***Plaid*** payment, it will communicate with ***Plaid*** and confirm that the bank transfer has properly taken place.
    - **<TO COMPLETE when the *Plaid* functionality is enabled>**

This is all we will describe in this section. For more details about the ***Plaid*** services refer to section **External Vendors**

* + If it is a ***Stripe*** payment, it will communicate with ***Stripe*** and confirm that the credit card payment has successfully taken place.
    - Different steps will be taken that are necessary in the Stripe environment, like
      * Fetch escrow account for the selected company
      * Get the ***setup intent*** created in the **Invest Now** popup with ***Stripe***
      * Fetch the ***payment intent*** and validate that it is in “***succeeded***” status
      * Update the settlement process (**primary\_investment\_request\_settlements table)** and its detail (***primary\_investment\_request\_settlement\_details table***) with the result of the previous operation.
  + If it detects any other payment method, the fiat settlement will be rejected as there are not any other method at the time of writing this document.
* After all the investments requests affected by the input json are processed, some of them will have been successfully processed and some of them will have found some sort of issue/error during the payment. A json will be returned specifying for every investment request, the result of its fiat payment confirmation.



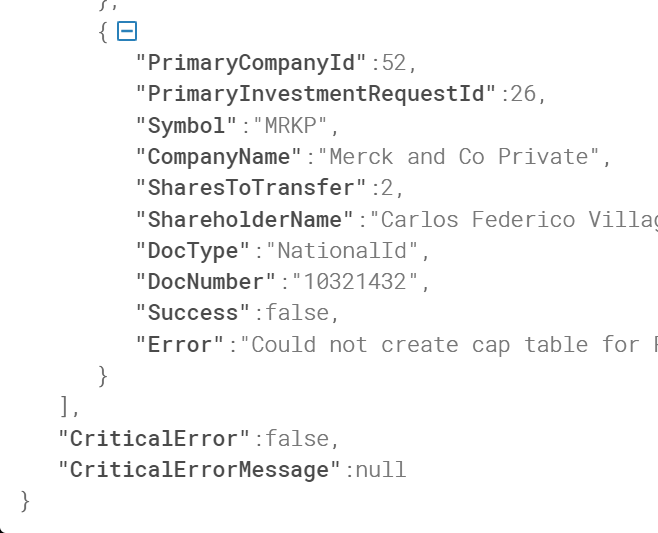
GetCapTableToTransferShares service

This is the third step of the settlement process. Basically, after all the fiat has been transferred, the system will receive a request to create a list of all the shares that have to be transferred to the new owners. In order to achieve this, the system will return a json that will describe for every row:

* The primary investment request id affected
* The new shareholder personal information
* The security symbol that has to be transferred
* The number of shares to transfer
* If there was any error processing current primary investment request

In this context, it is important to remark the following concepts

* Input Json: This is a json that will tell this service to clear a given number or range of investment requests. Refer to the *OnFiatLegPaymentRequest* service to see how all the parameters of the input json work.
* Actions taken:
  + The service will fetch all the existing investment requests that are in the ***APPROVED*** status for the provided filters (see Input Json) at table ***primary\_id\_verification*** related to the ***primary\_investment\_request***)
  + It will fetch all the settlement records in the **primary\_investment\_request\_settlements** tablefor the previous *primary investment requests* found
  + It will validate that the settlement is in ***FIAT\_TRANSFER\_CONFIRMED***  status (***status*** field in **primary\_investment\_request\_settlements** table)
  + Then it will proceed to create a Json, that will have the following fields



The fields are

* Primary Company Id: field ***id*** , table ***primary\_company***
* Primary Investment Req. Id: field ***id***, table ***primary\_investment\_request***
* Symbol : field **symbol** , table ***primary\_company***
* Company Name : field **name** , table ***primary\_company***
* Shares To Transfer : field **investment\_shares** , table ***primary\_investment\_request***
* Shareholder Name: field ***first\_name and last\_name*** , table ***primary\_investor***
* Doc Type: field ***doc\_type*** , table ***primary\_investor***
* Doc Number: field ***doc\_number*** , table ***primary\_investor***
* Success : true/false if there was any error in the record generation process
* Error: If there was an error, this will be the error description

All the previous records will be returned in a json with the following structure



And a new record will be inserted in tables ***primary\_cap\_table\_settlement*** and ***primary\_cap\_table\_settlement\_details.***

This is because the settlement record will be marked as ***SHARES\_TRANSFER\_REQUESTED*** in the **primary\_investment\_request\_settlements** table and this third step will no longer be able to be triggered again without getting an error. So, if for some reason the cap table is lost, or it was to be retrieved again, it will have to be fetch from the ***primary\_cap\_table\_settlement*** and ***primary\_cap\_table\_settlement\_details*** tables***.* This functionality is not available at the time of writing this document.**

OnConfirmSharesTransfer service

This is the fourth and final step of the settlement process. Basically, after all the shares have been transferred (cap table file is generated), the system will receive a request to confirm that the settlement process has finally arrived to and end and that all the shares have been transferred to the new owners.

In this context, it is important to remark the following concepts

* Input Json: This is a json that will tell this service to clear a given number or range of investment requests. Refer to the *OnFiatLegPaymentRequest* service to see how all the parameters of the input json work.
* Actions taken:
  + The service will fetch all the existing investment requests that are in the ***APPROVED*** status for the provided filters (see Input Json) at table ***primary\_id\_verification*** related to the ***primary\_investment\_request***)
  + It will fetch all the settlement records in the **primary\_investment\_request\_settlements** tablefor the previous *primary investment requests* found
  + It will validate that the settlement is in ***SHARES\_TRANSFER\_REQUESTED***  status (***status*** field in **primary\_investment\_request\_settlements** table)
  + Then it will proceed to mark all the settlements as ***Settled***.
* After all the investments requests affected by the input json are processed, some of them will have been successfully processed and some of them will have found some sort of issue/error during the payment. A json will be returned specifying for every investment request, the result of its settlement confirmation.



**II.E.v- Jumio ID Verification**

This service is responsible of validating with Jumio that a given application for a given shareholders has been approved or not.

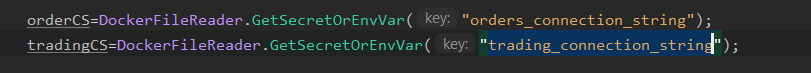
It consists of a thread that will

* Fetch all the investment request in the ***CREATED*** status (table ***primary\_id\_verification***)
* It will update the ***update\_verification\_stamptamp*** fields of the previous entity so as not to overload the Jumio server with too many requests (max 10 verifications for the first 24 hours, then one per day).
* If the application is approved, the primary investment request is marked as ***APPROVED***, and the settlement record is marked as ***KYC\_APPROVED***. Then the settlement dashboard will allow this investment request to request the money to be transferred for this investment request.
* If the application is rejected, the primary investment request is marked as ***REJECTED*** , and the settlement record is marked as ***KCY\_REJECTED***. **No more actions can be taken for this investment request at the time of writing these lines.**
* The previous process is repeated every 2 seconds.

**II.E.vi- Amazon Secrets**

To properly set up the net core services, the ***app.config*** file can have different settings that will be read in the application set up. However, some settings is not safe to have them in these config files. These settings are the connection string and they will be stored in the AWS secrets which is like an environmental variable that the Amazon servers provide.

So, all the net core service will be able to access the ***orders\_connection\_string*** and ***trading\_connection\_string*** where the different settings to access both catalogs lye.



**II.F - Data Synchronization**

The ATS is a platform that by its very same definition, needs to synchronize and transfer/receive information from/to different agents. This service has several endpoints whose sole purpose is to keep all the needed information fully synchronized.

The existing endpoints are:

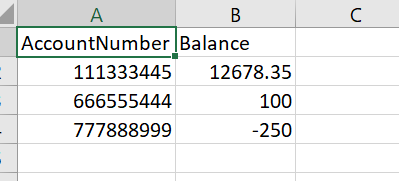
* **Management/PositionsSynchronization:** Used to synchronize with the different transfer agents all the existing positions in their portfolios and persist this in the positions tables. This table will be used to validate the existing balance when creating a sell order. This synchronization is implemented as a service which is called after logging in to the platform.

The Balance Synchronization will be done through the existing job **PiermontBalanceSyncApp.**

This app will connect to the Piermont SFTP server and download a csv file that will have for every account the existing balance.

This is an app that will run as a job at specific points in time during the day.

The downloaded file format must have the following format:



The SFTP server url and password can be retrieved from the table *external\_servers*, key *SFT\_ACCOUNT\_BALANCE\_SYNC\_PIERMONT*.

**Trading**

CLOBOrderBookApp

This is one of the most important applications as it is the one that implements the logic that allows the system to run a full real time central limit order book (CLOB).

It consists of server threads that work concurrently to guarantee that all the orders that arrive to the exchange are properly matched and published to whoever subscribes to the proper endpoints.

The app consists of the following threads

* Persistence
  + *PersistTradeThread*: Responsible of persisting all the matched trades
  + *PersistOrderThread*: Responsible of persisting all the changes that happen to an order through their lifetime
* Publish
  + *PublishEntryThread*: Responsible of persisting all the changes in the order book to the subscribed clients
* Refresh
  + *UpdateOrdersThread*: Updates the order book with all the new orders that are inserted to the database.
  + *SecurityUpdaterThread*: Updates the security list all the newly created securities.
* Trading
  + *OrdeMatcherThread:* This is the main logic, which is responsible of matching buy and sell orders prioritizing them on a price-time basis as it is usually done in a central limit order book (CLOB).

**II.G -Mocks/Testing**

Some other applications are needing for testing purposes to simulate a real trading environment, having real historical prices and active and tradable orders in the order books so that automated , demos and manual tests are easier to implement.

HistoricalPricesLoaderMock

Responsible of populating fake prices the table *jhi\_historical\_prices* which has all the daily historical prices for a given security. This way all the market data indicators and graphs can have realistic and testable data

OrderBookLoaderMock

Responsible of populating fake orders to the *clob\_orders* table so that the order book is always filled and there is enough liquidity for tests and demos. It’s worth to mention that these orders will be actually tradable, interacting with the orders created by the different shareholders.

**II.H –Most popular issues**

This section is thought describe a series of well known issues that took place during the application building life cycle and the tasks that were needed to properly fix them.

II.H.i - CORS issue

Context: The browser throwing CORS error in different interactions between the UI and some backend services, even after going through some backend services.

Solution:

The CORS issue was not actually a CORS issue. The ***preflight check*** could not be performed because the ***HTTP OPTIONS*** call failed. So it was a backend communication failure.  
 The ***Load Balancer (LB)*** for the secondary service was not properly redirecting traffic to the API. Hence all the API calls failed, including the ***HTTP OPTIONS*** needed for CORS.  
 Once the ***LB*** was fixed and properly configured, all the traffic for secondary service was received and the service become healthy again.

II.H.ii - Jumio Integration

Context: This issue refers to the issues experienced when setting up the Jumio identity verification iframe for the Primary Market

The issue with Jumio, the one related to the embedded workflow skipping the ***xDevice*** selection was due to not comply with the integration requirements on our end. In particular the requirement of using HTTPS. Our integration was using http, and even though it worked, it was not able to perform all of the steps in the workflow. Once we switched to ***https*** the problem was solved.

**II.I –External Vendors**

As was mentioned in section I , the ATS is an extremely integrated app. It interacts with different vendors that provide different services like capital tables management, bank account management, credit card payment, KYC validation, etc.

Every vendor has its own technical details that should be kept in consideration when troubleshooting or building any functionality. Following we provide the main concepts of every vendor.

II.I.i – Jumio services

Purpose:

* Used in the *Primary Market* (besides Solidus intervention) for
  + Extract and validate new investor´s documentation
  + Face match existing selfie/picture with the documentation provided
  + Run a whole Know Your Customer procedure for the new investor
* When a new *Primary Investment Request* takes place
  + The new investor picture and documentation is provided to ***Jumio*** and in exchange, the system receives a scan reference code that is persisted in table *primary\_id\_verification*.
  + Then, a new service ***Jumio ID Verification*** invokes Jumio service <https://auth.amer-1.jumio.ai/oauth2/token> and download the application status.
  + **At the time of writing this document no other documentation or information is downloaded from *Jumio* besides the application status**
  + Then the primary investment request is market as
    - KYC Approved: The settlement can be started
    - KYC Rejected: The settlement cannot take place
  + If approved, the settlement process can now begin

Management Tables:

* Refer to ***II.D – Database***, ***jumio\_settings*** table

Tokens used:

* Every time a new investment request arrives, there will be a token which is the one that will be used by the ***Jumio ID Verification*** process to confirm that the new investor’s KYC has been approved or not.
* This token will be link to the primary investor and the investment request, and will lie in table *primary\_id\_verification*, column *jumio\_id\_scan\_reference*.

Reference Links:

* You can access al the documentation about Jumio services in the following link
  + <https://customer-portal.netverify.com/app/#/verifications>
* You can access Jumio dashboard with the right credentials in the following link
  + <https://customer-portal.netverify.com/#/login?returnUrl=verifications>
* You can get more information about Jumio ID Verification service in the following section
  + *II.E.v-* **II.E.iv- Payment Rails**
* You can check about the Jumio picture upload troubleshooting in the following section
  + *II.F.ii – Jumio integration*

II.I.ii – Plaid services

Purpose:

* Provide banking balance and account management functionalities
* Provide bank transfers functionalities (under application)

Management Tables:

* Refer to ***II.D – Database***, ***plaid\_settings*** table

Tokens used:

* Secondary Market: When synchronizing a bank account an ***access token*** and ***item id*** will be provided by ***Plaid*** and stored in table ***plaid\_credentials*** that will be later used for
  + Transfer fund requests.
  + Trade settlement
* Primary Market: When receiving an investment request and synchronizing a bank account an ***access\_token*** and an ***account\_id*** will be provided by ***Plaid*** and stored in table ***primary\_plaid\_credentials*** that will be later used for
  + Primary Investment Request Settlement

Reference Links:

* You can access the Plaid dashboard at <https://dashboard.plaid.com/overview>
* There you can get all the client and secrets to populate the management tables

II.I.iv – Stripe services

Purpose:

* Provide credit card payment functionalities
* It is important to remark that Stripe also provides other payment methods, which have not been implemented at the time of writing these lines.

Management Tables:

* Refer to ***II.D – Database***, ***stripe\_settings*** table

Tokens used:

* Credit card payments are a sophisticated process that imply several steps
  + First, when populating the credit card, an embedded iframe will allow the user to type the credit card numbers. In return, a *setup\_intent* will be provided.
  + This setup\_intent will arrive to the payment rails attached to the payment request and it will be stored in table *primary\_stripe\_credentials*.
  + Then, when clearing this investment request the payment rails will use this *setup\_intent* to create a *payment\_intent* and trigger the payment.
  + Later, in step two of the settlement process, the system will confirm with Stripe endpoint that this payment intent was executed successfully.

Reference Links:

* You can read about the whole interaction between the *setup intents* and the *payment intents* and all the endpoints involved in the following link
  + https://stripe.com/docs

**III – Functional Workflows**

**III.A- Onboarding**

As was mentioned before, as the onboarding process is highly couple to every transfer agent, there will be an specific document describing all the technical details related to the to the interaction with a given transfer agent. For example, the document that describes the interaction with Kore Con X is ***Rialto KCX Solidus integration - Functional specs v1.x.***

However, there are a lot of functional concepts that describe everything that take place when onboarding a new shareholder. All these concepts apply to all the shareholders and no transfer agents related concepts will be developed in these paragraphs.

Basically there will be two tables that will control everything that takes places when onboarding a new shareholder: *onboarding status* and *onboarding details.*

The onboarding status registers all the different onboardings that take place in the platform. The main key for every onboarding process will be shareholder’s login, but different actions could be taken if an already existing tax Id is detected in the platform.

In this context we will have 2 kind of onboarding’s at the time of writing this document

* Transfer agent started the onboarding
* KYC agent (Solidus) started the onboarding

Transfer agent started the onboarding

When in a situation of a transfer agent starting an onboarding process, the following actions will take place

* An ***onboarding status*** will be created for that shareholder and marked as *INITIATED*
  + If a previously existing onboarding existed, it will be reset and taken back to the *INITIATED* status
  + An ***onboarding detail*** record will be created with all the different steps that take place for that onboarding from now on.
* If the shareholder already exists by login and taxId
  + If these shareholders are not the same person, then an error will be through until is corrected
    - The ***onboarding status*** will be marked as *FAILED*and returned to the shareholder
  + If they are the same person, the onboarding procedure will be assigned to this shareholder.
* The shareholder will be created/updated in the platform and sent to **Solidus** for KYC validations
  + If the communication with **Solidus** fails, then the whole onboarding process will be considered rejected (***onboarding status*** marked as *REJECTED*) and the proper message will be sent to the invoking transfer agent.
* The account creation package will be created and sent to **Piermont**
  + If this fails, the onboarding will go on, but this will be informed through an ***onboarding detail*** record and the account will have to be manually created later
* If everything went ok, the ***onboarding status*** will be market as *STARTED*
* The next step will be to receive the **Plaid** credentials from **Solidus** through the *Plaid Credentials Service*.
  + If there is some issue when receiving them, this will be informed as an ***onboarding detail*** and will have to be manually addressed later.
  + The ***onboarding status*** will stay as *STARTED* as usual
* The final step will be to receive the application approval message from **Solidus**.
  + The previously existing onboarding status will be detected and it will be this application the one that will be updated.
  + If the application is approved, the ***onboarding status*** will be marked as *APPROVED*. Otherwise it will be marked as *REJECTED.*
  + If approved, the shareholder’s password will be updated by the password populated in **Solidus** onboarding page.
  + The shareholder can now log in to the platform and start trading, requesting funds to be transferred, etc..
  + If there was any kind of problem creating the **Piermont** account or receiving **Plaid** tokens, this will have to be addressed manually

**<TODO: update how the different documents will be received and re transmitted to Piermont>**

KYC agent (Solidus) started the onboarding

In this process, the onboarding process is directly triggered and managed through the **Solidus** platform and the application services start doing their job directly at the point where the application approval or rejection is triggered.

After receiving an application approval/rejection, the following steps will take place

* If the application is approved
  + If the ***onboarding status*** did not exist it will be created as *STARTED*. Otherwise it will be updated to the *STARTED* status.
  + If it is approved the shareholder will be created or updated with all his personal data received from **Solidus.**
  + The shareholder’s password will be updated with the one received from **Solidus**
  + If there is any **Plaid** token unlinked that was previously received through the *Plaid Credentials Service*, it will be linked to the onboarded shareholder’s account
  + The **Piermont** account creation package will be created and sent to the **Piermont’s** SFTP
    - If this fails, the onboarding will continue as usual, but a new ***onboarding detail*** record will be created informing this situation
    - The **Piermont** account creation and population will have to be addressed later.
  + Finally, the **onboarding status** is marked as *APPROVED* and the shareholder can start using the platform.
    - It will be able to start trading and fund his account by the time account creation confirmation package is received from **Piermont.**
* If the application is rejected
  + The *STARTED* ***onboarding status***, previously existing or not, is marked as *REJECTED*.
  + If the shareholder previously existed, no update is made to his personal information.

**IV – Development process**

Software development process is a whole topic on its own and it is known that the mainstream methodology in today’s projects is the famous ***Agile Methodologies***, more specifically ***Scrum***.

It is not the purpose of this document to go through all the different tools and artifacts present in this methodology, but we can say that we have the traditional activities of this framework

* Daily Meeting
* Sprint plannings
* Sprint demos
* Sprint retrospectives
* 2 week sprints

However, the goal of this section is to describe how we try to implement this methodology in a regionally distributed team with marked different skills and knowledge.

Analysis process

I will talk in first person in this process, as a team leader I consider that the main component of a good analysis process is to get to know the problem that you are dealing with.

Therefore, we have

#1 – The now very old cascade methodologies where the process goes from conception, analysis, all the way until testing

#2 - The process described in the *Agile Framework*, where the previous process described in step #1 is split into a whole set of different micro projects called sprints, each one with its own development cycle.

As was mentioned, we feel much closer to #2, but the following particularities:

* The analysis process is a ***2-step process*** where the product owners (***Rialto*** team) communicate a high-level feature to the team leader who is responsible, based on his experience in the financial markets, to design the functionality that best fits that high-level description.
* As this is an iterative process, part of the analysis will be done through demos. This demos will take different formats
  + One demo will be done with the development team so that they get to know what they are building, their strengths , weakness, opportunities to improve what has been developed and what is more important, they get to know what has been built, and what it is useful for. They stop seeing the product as “grids and buttons” developed by some tool of their expertise to get an idea, little by little, of the problem they have to solve. As it is said, if you have a hammer, you will see nails everywhere, but it is here where they will start seeing nails, screws or whatever the project needs instead of what their tool needs.
  + However, the other part of this process will be focuses on the ***Rialto’s*** management team. That is why we implemented after any development process, a series of meetings called walkthroughs, where we try to use the product, real daily scenarios where the user gets to know not only what the apps does , but also how it does it. There is no better way to know how it feels using a given tool, than actually using it and facing real functional situations. All that was described in the previous paragraph might seem like part of the testing process, but it is here where the best ideas and improvements are born in a real iterative process.

Development process

As was mentioned there are 3 basic technologies

* Angular for the UI developments
* Java for the backend developments
* .net Core for the isolated functional developments

Therefore, once we have the high-level description of the desired functionality buy the Rialto management team, a bottom up design by the development team, we have to take the following decisions

* Technology to build it
* New feature architecture
* Interaction with the UI if needed

Here we usually take a risk-centered approach. We believe that the ***Paretto law*** applies very strongly to all the software development processes. This means that any projects whose length is no more than 6 months length will have one ***core functionality*** where all the risks lie, and ***additional functionalities*** that strongly depend of this core functionality or at least that orbit around it. If this core functionality is strongly built and validated, we will have cleared most of the project´s risk. Even if later, these additional functionalities start to experience some delays or hurdles, they can quickly be controlled, isolated, solved, and a situation where general failures, or one failure escalating to multiple failures have much less probabilities to happen.

In this context, all the following functionalities where considered ***core functionalities***

* CLOB Engine
  + Multiple concurrent activities critical in what is related to performance and robustness
  + Multiple layer of validations with a strong need for escalation
  + Multiple layers of communication with external apps
* Onboarding
  + Sophisticated encryption and decryption mechanism of sensitive data
  + Interactions with payment agents (Plaid), KYC validation vendors
  + Consistent shareholder situation management
    - Provide communication for failed onboarding
    - Frictionless experience for a successful onboarding
* Payment Rails
  + Interactions with different payment vendors encrypting and decrypting sensitive data
  + Robust logging and documentation
  + Robust and clear interfaces
  + Grant investments consistency and avoid double accountability scenarios

For all the previous reasons, all the previous functionalities where developed in ***net core*** decoupled services, building around them the following functionalities (that we call ***satellite functionalities***) using ***Angular*** and ***Java***

* Order books
* Financial indicators
* Trading Settlement processes
* Primary Market Settlement Dashboards
* Investment management Screens

Testing process

Given all the previous descriptions, it will the tech lead, the very same that coded the core functionalities, the one responsible for testing all the satellite functionalities.

So the testing activities will have the following structure

* The ***core functionalities*** are developed by the high-seniority team members a deep knowledge of the problem (*problem-center profiles*) to solve and the technologies used
* The ***satellite functionalities*** are tested by those team members
* The ***satellite functionalities*** are then automated to assure its stability and robustness on the long run.
* During the daily operations, the senior team members, validate that the automation stack ran properly and fix any failure that might have existed
* After every ***satellite functionality*** has been developed, manually tested and deployed, the automation process should immediately start. It will be responsibility of the senior members to grant the existing functionality the good enough amount of quality until the automation is finished.

Architecture

Refer to ***Net Core - Java- Architectural and Deployment diagrams v1.x.docx***

Deployment process

A deployment process will consist of the following steps

* Deploy the latest features in dev
  + Test all these features
* Promote all the latest dev releases to staging
  + Manually Compare tables
  + Manually Compare stored procedures
  + Manually compare ***system\_variables*** table
  + Manually compare ***external\_servers*** table
  + Manually compare ***external\_services*** table
  + Manually compare all the ***settings*** tables
  + Turn off *Liquidity Providers*
  + Turn on *Application Approva*l service
  + Turn on *CLOB Engine* app
  + Turn on *Banking Services* svc
  + Turn on *Jumio ID Verification* svc
  + Turn on *Plaid Credentials* svc
  + Turn on Positions Sync svc
  + Turn on Liquidity Providers
  + Create dummy *Primary Application* service (manually approve it)
  + Subscribe *Order Book @ CLOB Engine* app
  + Invoke simple service at *Banking Services*
  + Settle dummy application with test clients
  + Run all the automation algorithms if existing
* Promote to UAT
  + Repeat the previous process
* Promote to PROD
  + Repeat the previous process extensively

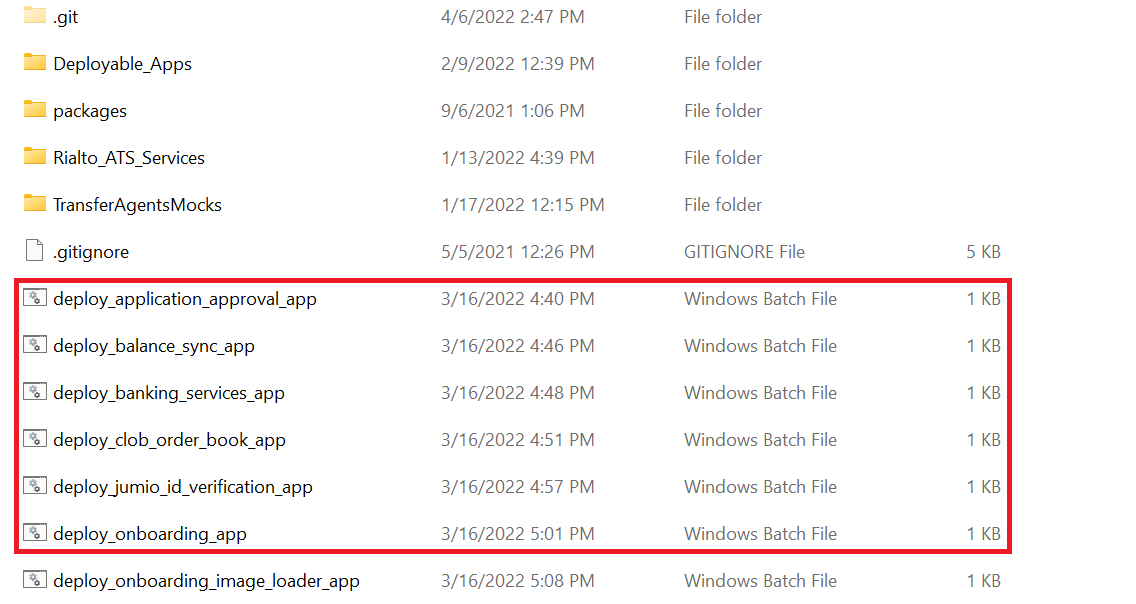
It is important to remark that

* All the build and deployment is done automatically in all the lower environments (Development , staging and UAT) after pushing a commit
* For some naming convention decision, UAT is the pre productive environment and not staging as you might find in other environments
* The productive deployment has to be done manually, specifically requesting the deployment to the Devops team

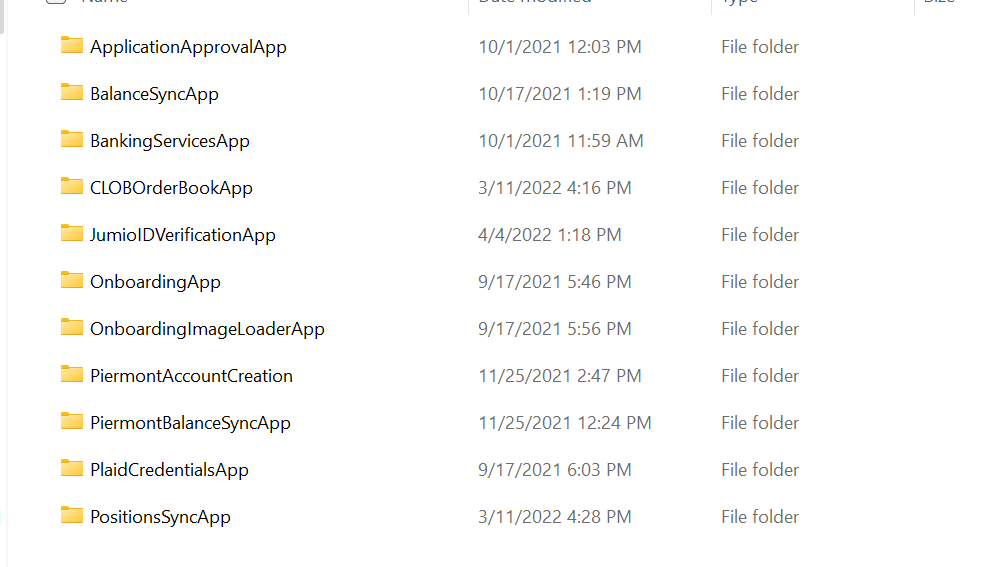
In what is related to the net core world, aligned to what was described as a *core to satellite* strategy, there will be 2 repositories

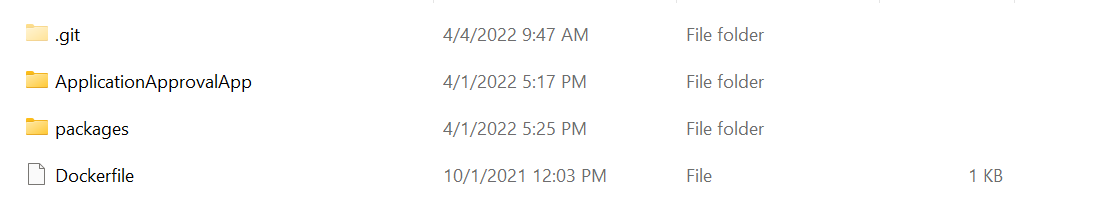
1. <https://github.com/TradeHelmInc/RialtoATSNetCoreFwk>
   1. This will repository will have all the code, with the different layers, logics, calculation and test clients. This code has to be pulled if you want to analyze what the net core repository is made of (code, architecture, entities, etc.).
2. There we will be different satellite projects that
   1. The will have a ***package*** folder where all the core project assemblies will be stored
   2. A script will be in charge of
      1. Building the ***main core project***
      2. Copy all the output assemblies to the ***packages*** folder of the built ***satellite*** project
      3. Replacing the ***Program.cs*** from the ***core project*** (see #1) to the ***satellite project*** (only piece of code in the satellite project)

For more information follow, analyze all the following scripts described in step 2.b



Once step 2.b has been run, you can commit and push the affected ***satellite project*** in the ***\Deployable\_Apps*** folder.

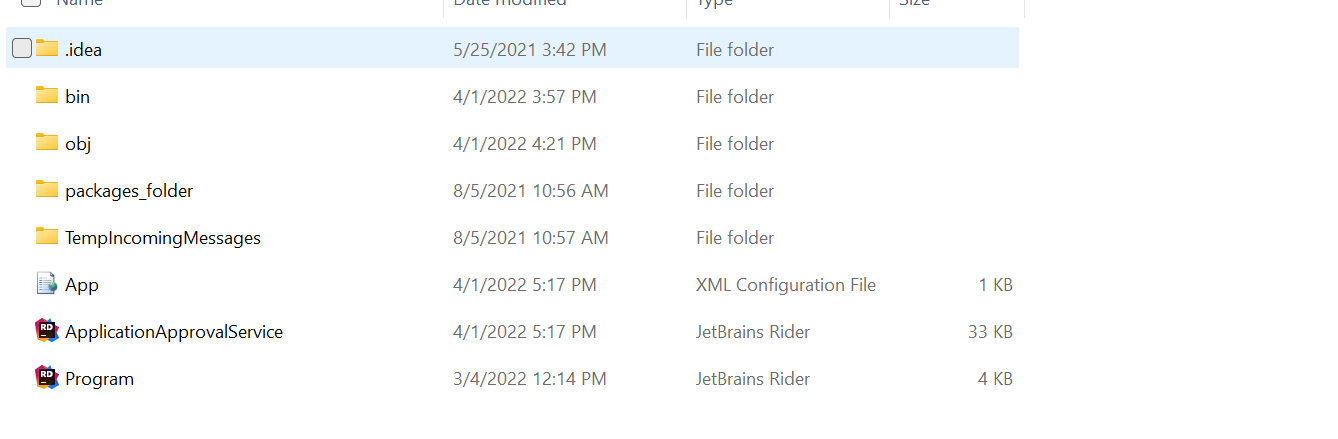


Every satellite project has a very simple structure of a ***packages*** folder 

An inside the ***SatelliteProjectApp*** folder you will have a net core solution whose unique function will be

+ Reference the assemblies in the packages folder.

+ Read all the needed settings from configuration files or Amazon secrets.  
 + Start the needed class/service depending what the satellite project is.



* The configuration files (See how we can populate connection strings in it, but in the Amazon environment we should use the secrets through the ***UseSecrets=true*** variable)



* The bootstrap file. A simple process of reading all the necessary settings, starting the needed app (like ***ManagementService***), then logging and sleeping as necessary.

