Frontend - Technical Documentation

Development System Requirements

Software

For the development of the application we used 2 particular Operating Systems namely; MacOS and Linux (Ubuntu).
Windows is not recommended as running a local server and the test suites will cause trouble. A lot of these issues can't be solved.

The following software is required for the development of the application.

- NodeJS
- Meteor
- An editor capable of recognizing JavaScript

Hardware

For running a local server not much is required, the most basic hardware is able to do this although it might be slightly taxing when using big database queries.
To speed up the process of rebuilding the server a faster system would be prefered.

The following requirements are just a recommendation for a smooth development process:

- Intel i5 or similar
- 8-16GB of RAM
- Fast HDD or SSD with at least 3GB of space left
- A screen with a resolution that is 1920x1080 or more

Code Structure

This is the way the team discussed and decided to program and structure the software. For a team, a good standard helps maintainability and keeps everything clear when others read it.

Naming Convention

Naming conversion are the set of rules when naming folder, files, functions, etc. These rules were set so that other developer can understand the code just by looking at the name instead of reading the whole code and try to understand it.

- We do not use abbreviations in names, ex use “language” instead of “lang”.
- Make names as self-describing as possible (comments should never be needed for names).

	Style	Example	Remarks
Folder	Dasherized	important-folder	-
File	Dasherized	good-file-name.js	File type or content doesn’t matter.
Routes	Dasherized	my-path-name	-
Templates	Lower Camel case	remarkableTemplate	-
Variable	Lower Camel case	someVariable	Boolean variables should start with “is” or “has”.
Function	Lower Camel case	niceFunction	Don’t let function start with “get”, “has”, etc edit any states.
Function Constructor	Upper Camel case	MyAwesomeClass	-
CSS classes/ids	Dasherized	my-awesome-button	-

Folder System

In this section you will find the most relevant folders in the entire application.

- **attainenroll**: Sales application main folder, it contains all the MeteorJs code.
  - **client**
    - **lib**
      - **components**: Generic components that are used multiple times.
    - **stylesheets**: Code written in LESS about the application styling.
    - **views**: Blaze templates related to the application main views.
  - **imports**
    - **client**
    - **lib**
      - **fixtures**: Model builders and factories.
    - **server**: Modules like email actions and invoice state machine.
  - **lib**
    - **collections**: Collections that are used in client and server side.
    - **locale**: Translations of the application.
  - **packages** Modified packages
  - **private**
    - **emails**: html templates for the emails.
      - `en` for English emails
      - `ja` for Japanese emails
      - `ko` for Korean emails
      - `zh` for Chinese emails
  - **public**: Application public resources like agreements pdf files.
  - **server**: Meteor calls, publications, etc.
    - **collections**: Collections that are only available in server side, like jobs collection.
    - **emails:** Server side rendering for complied email templates
    - **lib:** Server related codes.
    - **workers:** Codes for all the server side workers
  - **test**
    - **end-to-end:** Tests written with the Gherkin language and cucumber. Most tests are in subfolders separated by features.
      - **_support:** Tests helpers and common resources
      - **unit:** Gherkin and cucumber unit styles tests.
      - **integration:** Gherkin and cucumber integration styles tests.
    - **integration**: Integration tests (written with mocha).
    - **unit**: Unit tests **(**written with mocha).
- **bin**: Scripts used for execute, run tests and deploy the application.

- ➢ **deploy:** Scripts for deploying the application to test and production server.
- ➢ **run:** Scripts for running the application.
  - ■ "dev" are used to run the sales application on local machine.
- ➢ **test:** Scripts for running test of:
  - ■ End-to-end
  - ■ Integration
  - ■ Unit
- ➢ **env:** Contains environment settings for running application and tests.
- ➢ **db**: Scripts for importing the db from test environment to local storage.

- ❖ **config**: Settings files for the application. It contains sensitive information, such as server secret key, access tokens, so be careful not to put this folder in public place.
- ❖ **dump**: Test environments database dumps.
- ❖ **playhooks**: YAML configuration files, and encrypted credentials for mailgun, backend, and other services.
- ❖ **scripts**: Script to run on database. Scripts include:
  - ➢ **backfill**
  - ➢ **lookup**
  - ➢ **update**
  - ➢ **validation**
  - ➢ **exports**
  - ➢ **Development**
- ❖ **UX-Slides**: Application initial mockups, could be outdated.

Important Files

Client Router

File path: "/client/router.js"

This file contains all the routes that a user can access in the Sales application. Router defines the user permissions that is needed to access to each route and the Templates that are rendered.

Invoice State Machine

File path: "/imports/server/invoice-ticket-state-machine.js"

The expected ticket states and changes are defined by the Invoice state machine. All the information about the flows that a ticket could take are in this file.

API Endpoints

File path: “/server/router-server.js”

The API endpoints for the Sales application are defined in this file. Mostly they are used by the BackendApp or the AVVM in order to communicate special events.

System Boot

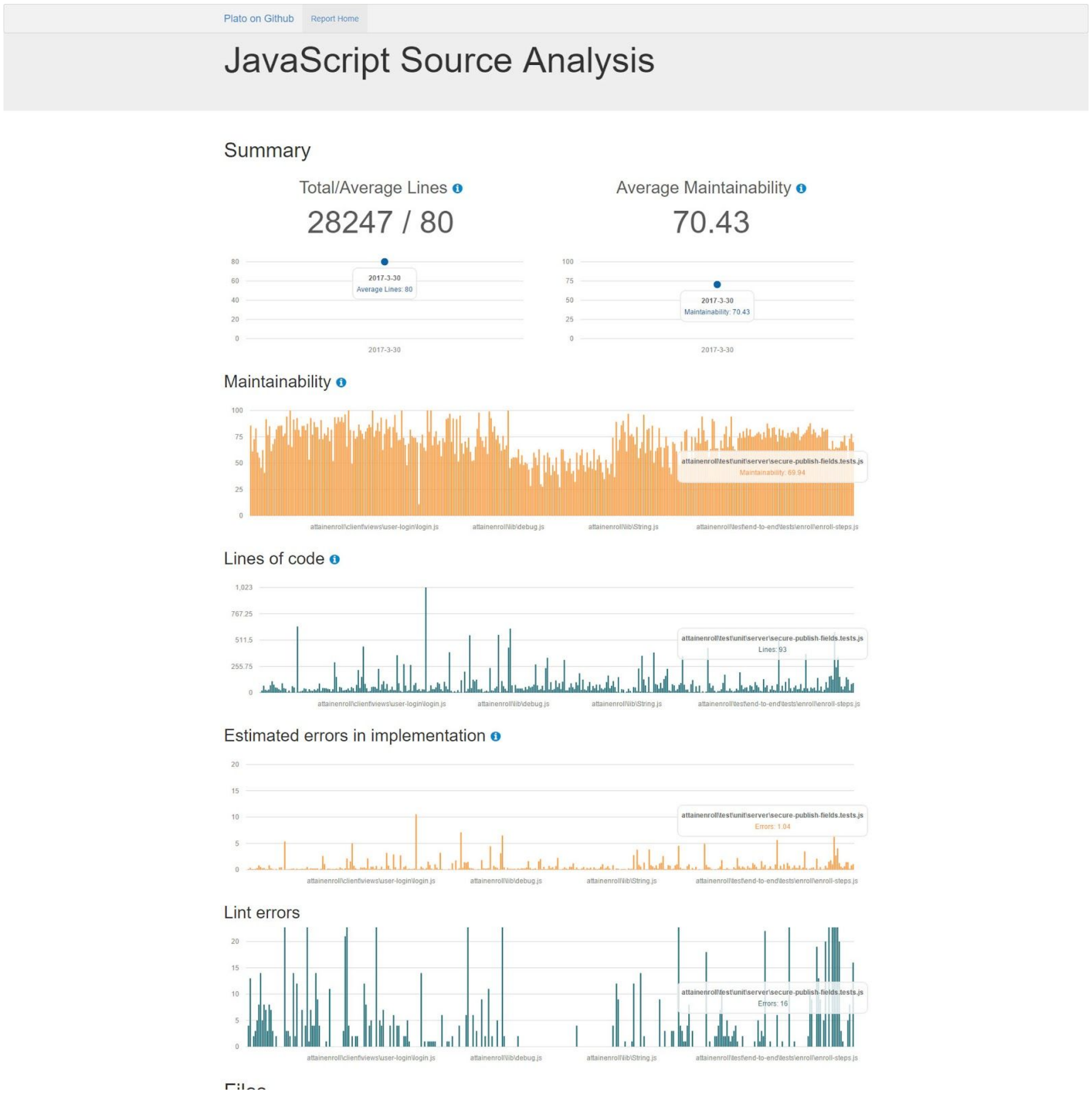
File path: “/server/init.js”

The System setup is in this file. Mainly the workers are triggered and the Logger is configured here.

Code Base Metrics

LOC

LOC = Lines of Code
NCLOC = Non Commented Lines of Code


Plato on Github
Report Home
80
60
40
20
0
2017-3-30
Average Lines: 80
100
75
50
25
0
2017-3-30
Maintainability: 70.43
attainenroll\client\views\user-login\login.js
attainenroll\lib\debug.js
attainenroll\lib\String.js
attainenroll\test\end-to-end\tests\enroll\enroll-steps.js
attainenroll\test\unit\server\secure-publish-fields.tests.js
Maintainability: 69.94
1,023
767.25
511.5
255.75
Lines: 93
Errors: 1.04
Errors: 16

JavaScript Source Analysis

Summary

Total/Average Lines
28247 / 80

Average Maintainability
70.43

Maintainability

Lines of code

Estimated errors in implementation

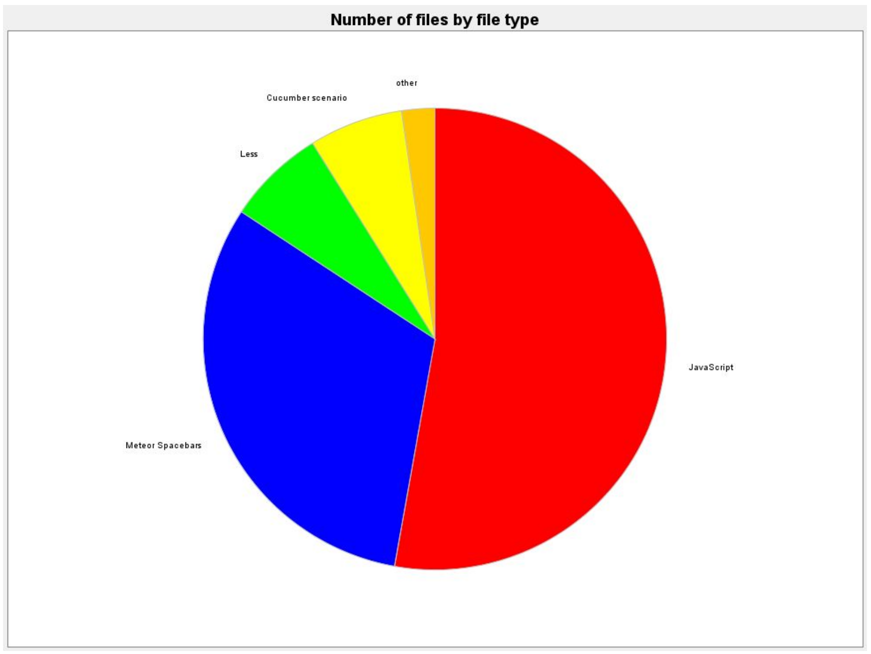
Lint errors

Files

File type	LOC	NCLOC
Cascading style sheet	52	50.0
Cucumber scenario	1660	1643
JavaScript	25365	24639
JSON	16	16
JSON Schema file	63	63
Less	3119	2643
Markdown language file	4	4
Meteor Spacebars	4186	4186
XML	151	146
Total	34628	33402
Average	4328.50	4175.25

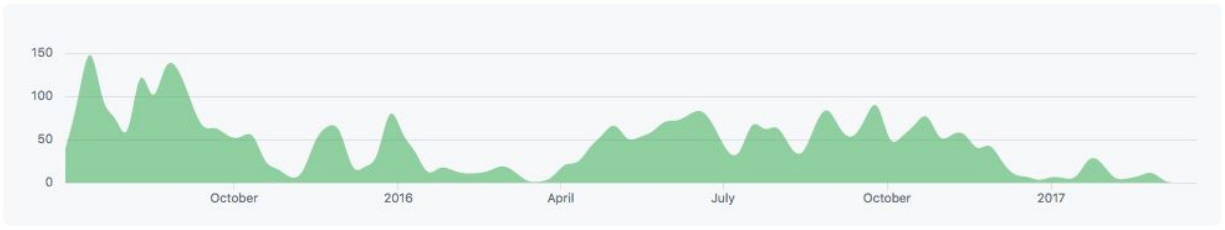
Files

File type	Amount
Cascading style sheet	3.0
Cucumber scenario	43.0
JavaScript	344.0
JSON	2.0
JSON Schema file	1.0
Less	45.0
Markdown language file	1.0
Meteor Spacebars	199.0
XML	8.0
Total	646.0
Average	80.75



Commits

As of Release Z4 we have 6,360 commits (including merge commits).



Code Contributors

For more details visit contributors section in github

References:

- Code contributors: https://github.com/input-output-hk/sales-frontend/graphs/contributors

Contributions per person (excluding merge commits)

Contributor	Commits		Additions		Deletions	
*****************	899	21.65%	59,928	21.07%	56,767	28.08%
*****************	360	8.67%	51,699	18.17%	26,220	12.97%
*****************	348	8.38%	20,517	7.21%	12,727	6.30%
*****************	318	7.66%	13,951	4.90%	11,601	5.74%
*****************	250	6.02%	6,446	2.27%	3,970	1.96%
*****************	226	5.44%	16,693	5.87%	6,007	2.97%
*****************	211	5.08%	9,606	3.38%	5,582	2.76%
*****************	209	5.03%	15,867	5.58%	5,672	2.81%
*****************	188	4.53%	11,347	3.99%	8,122	4.02%
*****************	184	4.43%	6,018	2.12%	9,042	4.47%
*****************	172	4.14%	9,784	3.44%	3,759	1.86%
*****************	168	4.05%	8,938	3.14%	5,881	2.91%
*****************	164	3.95%	43,642	15.34%	42,414	20.98%
*****************	102	2.46%	5,428	1.91%	1,270	0.63%
*****************	100	2.41%	348	0.12%	316	0.16%
*****************	66	1.59%	1,016	0.36%	430	0.21%
*****************	52	1.25%	698	0.25%	652	0.32%
*****************	44	1.06%	866	0.30%	325	0.16%
*****************	39	0.94%	553	0.19%	557	0.28%
*****************	38	0.92%	730	0.26%	708	0.35%
*****************	11	0.26%	401	0.14%	138	0.07%
*****************	3	0.07%	0	0.00%	0	0.00%
*****************	1	0.02%	7	0.00%	1	0.00%
Total	4153		284,483		202,161	

Test Cases

Test type	Amount
Unit	82
Integration	69
End-to-End	330

Development Process

Workflow

Github Projects, Trello

What is Trello

Trello is a collaboration tool that organizes your projects into boards. In one glance, Trello tells you what's being worked on, who's working on what, and where something is in a process. Here are some features that were used for our task management.

- Tags
  - Tags are used to indicate what the card is related to. If it's bug related, we add bug tag.
- Checklist
  - List of things that need to be done. This is used to check the progression of the work.
- Markdown
  - We used markdown to write down code snippets and file names.

References:

- Trello: https://trello.com/

What is Github Projects

Github Projects is a similar collaboration tool. But with Projects, you can manage work directly from your GitHub repositories. Create cards from Pull Requests, Issues, or Notes and organize them.

Although Trello and Github Projects have similarities, there are differences.

References:

- Github projects: https://help.github.com/articles/about-projects/

Trello

- Trello reactively shows if someone's editing the card. This means that team can check latest state of the card without making someone wait.
- Checklist are shown as a progress bar in the overview so it is easy for team to check the progression of the tasks.
- Trello has a mobile application so you can see the board using mobile phones, tablets.

Github Projects

- Syntax highlighting of code snippets.
- One system is a goal in itself.
- Can reference cards via commit.
- Only one person can edit a specific card at a time.
- No mobile application supported.

Why We Use Trello and Github Projects

We decided to use Trello as a board to use for planning meetings and GitHub Projects for task management of our development.

Github Projects is **not reactive** and only one person can create and edit task cards while the meeting is held. This means that most people had to wait until each card was finished, which was a waste of time.

Trello on the other hand is **reactive** and multiple people can work on same card at the same time. Using Trello has made our planning smoother since we did not have to wait for someone to finish editing the cards.

Task, Story Cards

In collaboration tools we have two types of cards, **story** and **task**.

Story

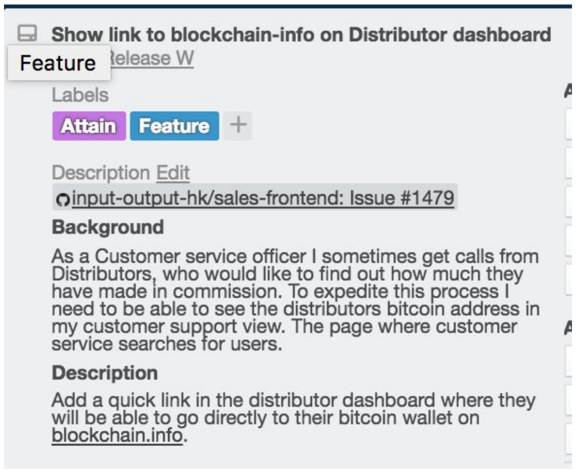
A “story” is a request from the client, development team or product owner which they want to discuss and implement to the application.

Story Format

Follow the story format so that the team and product owner can understand what the issue is.

- **Tag**
  - Add a tag of who has requested it. ex. client, product owner, development team.
  - Add what kind of issue it is such as feature, bug, chore, and analysis.
- **Title**
  - Explain what the issue is in a few words.
- **Background**
  - Describe what the issues are and why is this needed to be implemented.
  - If it is user interface related, paste in a picture as well.
- **Description**
  - Describe how it can be solved. Try to add a number of possible solutions.
  - Adding pros/cons would be helpful for the team to decide which solution they should implement.
- **Comments**
  - If there was any progression to the story, write them in the comment. Team members should not edit the story itself since we want the keep the timeline trackable.

Example:



Show link to blockchain-info on Distributor dashboard

Feature ~elease W

Labels

Attain Feature +

Description Edit

input-output-hk/sales-frontend: Issue #1479

Background

As a Customer service officer I sometimes get calls from Distributors, who would like to find out how much they have made in commission. To expedite this process I need to be able to see the distributors bitcoin address in my customer support view. The page where customer service searches for users.

Description

Add a quick link in the distributor dashboard where they will be able to go directly to their bitcoin wallet on blockchain.info.

Task

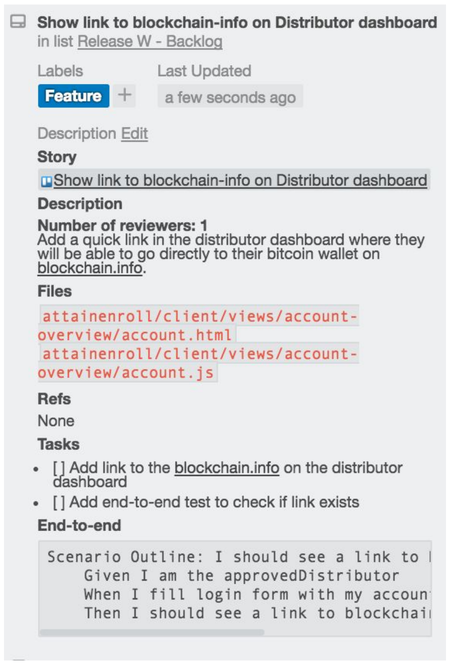
Task cards are used in development process. It is used so that the team understands what will be implemented and what they should be working on. Tasks are usually made from stories which are what the client, product owner, or development team requested.

Task cards contain more code related instructions than story cards. This is so that person taking the task would not waste their time figuring out what they actually have to do or how to implement it.

Format

- **Title**
    - Short description of what you want to accomplish.
- **Number of reviewers**
    - Number of persons to review this task. the number is decided on planning meeting.
- **Story**
    - Link URL to the Trello/Github story card.
- **Background**
    - Describe the issue and why is this task necessary.
- **Description**
    - Describe how the issue can be solved. If it is code related, write concrete instructions.
- **Files**
    - Link URL to the Github with develop branch or link may expire and we cannot see the files.
- **Refs**
    - Any reference that would help the team complete the tasks.
    - If any similar codes were used in the application, add a link to Github files.
- **Sub tasks**
    - Progression checklist of things that needs to be done to complete the task.
    - Each list is a mini task to complete the whole task. This will also be used to keep track of progression.
- **Acceptance**
    - Describe what kind of behavior is expected to see on the application after the task is complete.
    - It will be written in gherkin style so that development team can write and review them by end-to-end-test.
- **Comments**
    - If the person taking the task have any question of something he or she want to discuss with the team.
    - When you want someone else to take over the task, write what you've done and ask someone in the team to take over.

Example: This is an example of task card that implements new feature.



Show link to blockchain-info on Distributor dashboard
in list Release W - Backlog

Labels: Feature
Last Updated: a few seconds ago

Description Edit

Story

Show link to blockchain-info on Distributor dashboard

Description

Number of reviewers: 1
Add a quick link in the distributor dashboard where they will be able to go directly to their bitcoin wallet on blockchain.info.

Files

`attainenroll/client/views/account-overview/account.html`
`attainenroll/client/views/account-overview/account.js`

Refs

None

Tasks

- [] Add link to the blockchain.info on the distributor dashboard
- [] Add end-to-end test to check if link exists

End-to-end

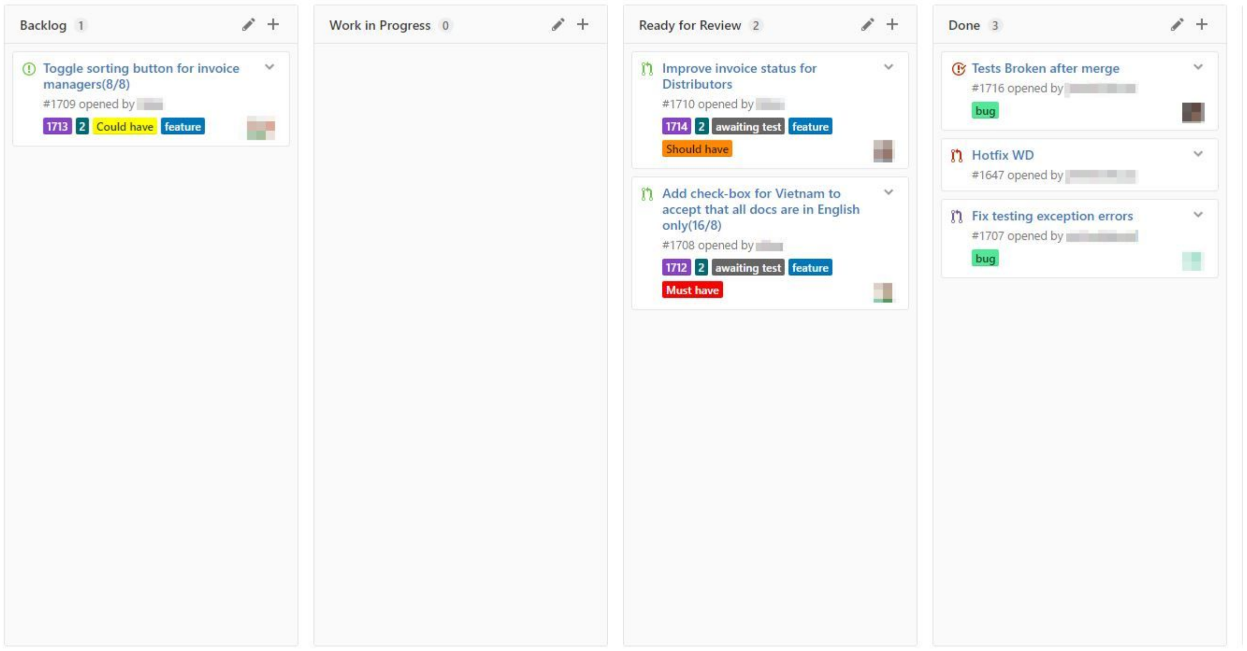
```
Scenario Outline: I should see a link to l
    Given I am the approvedDistributor
    When I fill login form with my accoun
    Then I should see a link to blockchai
```

Lanes

To keep track of development tasks, we have 4 lanes. Each lane represents a stage in the development process:

- **Backlog**
    - Not started, roughly prioritized, indeterminate completion date. Development team will select a task from this lane and move them to “work in progress”.
- **Work in Progress**
    - Once you've made a new branch or begun working on a task, you should move the card to the "Work in Progress" list.
- **Ready for review**:
    - When development is finished, move the card to "Ready for review" and paste a link to the pull request in the Trello card description.
- **Done**
    - Whenever you merge a pull request to develop, move the appropriate card to this list; you should only merge pull requests that belong in the next release.

Example:



Backlog 1

Toggle sorting button for invoice managers(8/8)
#1709 opened by [redacted]
1713 2 Could have feature

Work in Progress 0

Ready for Review 2

Improve invoice status for Distributors
#1710 opened by [redacted]
1714 2 awaiting test feature
Should have

Add check-box for Vietnam to accept that all docs are in English only(16/8)
#1708 opened by [redacted]
1712 2 awaiting test feature
Must have

Done 3

Tests Broken after merge
#1716 opened by [redacted]
bug

Hotfix WD
#1647 opened by [redacted]

Fix testing exception errors
#1707 opened by [redacted]
bug

MoSCoW

The MoSCoW method is a prioritization technique used for deciding which tasks are essential for the current release and which should not be prioritized. The labels used are:

Must have
  The release can't go out without this task being implemented.

Should have
  This task is not required to be in release, but implementation is highly recommended.

Could have
  If this task is finished, it can go in the release else save it for another one.

Won't have
  This task will not go in the release.

Scrum

What is Scrum

Scrum is an iterative and incremental agile software development framework for managing product development.

Main Features of Scrum

- A product owner creates a prioritized wish list called a product backlog.
- During sprint planning, the team pulls a small chunk from the top of that wish list, a sprint backlog, and decides how to implement those pieces.
- The team has a certain amount of time — a sprint (usually two to four weeks) — to complete its work, but it meets each day to assess its progress (daily Scrum).
- Along the way, the ScrumMaster keeps the team focused on its goal.
- At the end of the sprint, the work should be potentially shippable: ready to hand to a customer, put on a store shelf, or show to a stakeholder.
- The sprint ends with a sprint review and retrospective.
- As the next sprint begins, the team chooses another chunk of the product backlog and begins working again.

References:
- Scrum Alliance: https://www.scrumalliance.org/why-scrum
- Agile Software Development: https://en.wikipedia.org/wiki/Agile_software_development

Scrum in Our Project

Until Release Y we used Scrum approach for our releases. Before each sprint we did a planning meeting in which we created task cards for current user stories, prepared by the product owner, and estimated them. The typical sprint length was 2 weeks.

Cards were prioritized with the MoSCoW method (*Must have*, *Should have*, *Could have*, and *Won't have*). The releases continued until most of the important cards were in. We didn't do sprint reviews. The sprint was reviewed by the client and product owner. No sprint review between developers was held.

Every day we held two standup meetings. First with Osaka/Austria team and the second one with Austria/Argentina team. During these meetings we pointed out what we did that day, the day before and what blockers we had. We tried to keep the meetings as short as possible having a target of maximum 15 minutes.

Kanban

What is Kanban

Kanban is a popular framework for agile software development with a long history. The main concept is to split task cards in ‘To do’, ‘Doing’ and ‘Done’ lanes. The ‘Doing’ lane should have a limit of cards it can contain, this forces the team to finish tasks they have already started before grabbing new cards.

Kanban and Scrum share some concepts, but there are many differences. One of them is that Kanban uses a continuous flow instead of fixed length sprints, which means it is completely okay to add new cards at any time and any undone cards are automatically pushed to the next release. The task cards in backlog are sorted based on their priority.

The main benefits of Kanban over Scrum are the ease to do bugfixes, because in Scrum, if the fix is not a hotfix, you need to wait for the next release to get it out, which in some cases can take a long time. Kanban solves this problem by doing more frequent, flexible releases. Another benefit is doing ad-hoc tasks, since you can add/prioritize tasks in the middle of sprint.

Some drawbacks of Kanban are accumulation of technical debt. The reason for it is that in Scrum, for each release, you do a planning meeting as well as a sprint review meeting and a Sprint Retrospective Meeting. Having all those meetings usually leads to a more organized code structure. Kanban also has a problem with refactoring. Since Kanban doesn’t have a concept of code stop, it’s hard to find a good time when to do refactoring, without interfering with other development.

Kanban in Our Project

After Release Y we decided to stop implementing new features and focus on maintenance and bugfixes. At that point we started getting more ad-hoc tasks. Since in kanban it’s easier to change priorities for tasks and add new tasks in the middle of sprint, we deemed it more suitable to switch to the kanban approach.

We also did some modifications to the kanban method. For example, we continued with the planning meetings and daily standup meetings which are usually part of the Scrum workflow. Instead of going with the kanban method for prioritising cards, we stayed with the MoSCoW method. Also a limit of 5 cards was set for each lane.

We switched to weekly releases with approximately 3 story cards added in each of them and started using the continuous flow.

Weekly Development Meeting

Weekly developments meetings are held every Wednesday on 8PM Osaka time, and they are one of the most important parts in our development process, since they help us keep track of the following items:

- Sprint progression.
- What has been done.
- What needs to be finished.
- Any notices that needs to be notified to the team.
- Discuss topics that needs to be discussed with the team.

Every development team member was participating in the meeting. Our team was using Google Hangouts for the meeting and we recorded it by voice recorder. Recordings were posted in Slack's **dev-meeting-rec.** Basic agenda of the meeting was provided by the project manager.

Example:

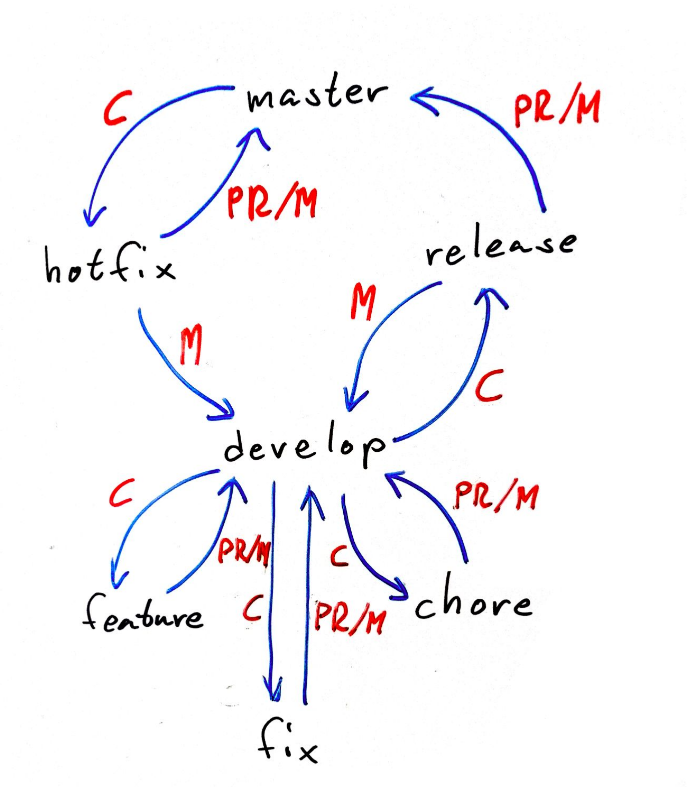
- State of development
    - Current state of development. What team has finished, what needs to be done.
- Must Haves
    - Clear out what need to be done as soon as possible.
- Deadlines
    - Deadline of our next release.
- Git
    - Git history needs improvement.
- Misc
    - We have discussed about certain tasks and why they are necessary.

Git

Gitflow

Git is not an improved CVS or SVN, Git is made for and thriving in branching and merging. Resolving merge conflicts is part of the workflow and git offers a multitude of merging strategies and conflict resolving tools for helping out.

So for a developer having a background in CVS or SVN it might be frightened when figuring that we apply something called GitFlow, which probably will feel like branching on steroids. Drop what you've learnt from CVS, SVN or similar tools and take a while to study up on how Git works, and this will be an easy ride.



References:

- GitFlow: http://nvie.com/posts/a-successful-git-branching-model/
- Git: https://git-scm.com/book/en/v2

Branches

- **master:** This is the branch containing the product, and every merge into this branch is a new version of the product.
- **develop:** The main development branch, no code should get directly committed onto this branch.
- **feature/…:** A feature branch is created for every new feature that has to be developed.
- **fix/…:** A fix branch is created for every non-critical bug that has been detected.
- **chore/…:** A chore branch is created if plain refactoring is needed.
- **release/x.x.x:** The release branch is the link between develop and master.
- **hotfix/…:** This branch type is only made if there is critical bugs that needs to be pushed directly to the product.

Commenting

Git Comments

Git comments are based upon Tgabbery's article "*A Note About Git Commit Messages*". If something is unclear or ambiguous, then fall back on that article as a second source of information.

The commit message guideline in bullet form:

- **The first line/headline**
  - Start with Trello card/ Github issue ID followed by the headline, separated by a dash.
  - **Max 50** character (including id, dash and space).
  - Written in imperative form(Fix instead of Fixed, Fixes, etc).
- **The second line**
  - *Always* empty.
- **The message body**
  - **Max 72** character wide.
  - Contains three sections each separated by an empty line.
    - Description
      - Title; Nature.
      - At *least* two sentences.
    - Location
      - Title; Location.
      - At least two sentences.
    - Solution
      - Title; Solution.
      - At *least* two sentences.

Example:

```
1234 - Fix bug in counter function

Description
-----------
The counter function increaseByOne increased the given value by two,
instead of the expected one. This caused the database to only contain
even numbered indexes.

Location
--------
The bug was located in increaseByOne, but the effect became apparent in
prepareNextRecord. Both functions are located in database_library.js.

Solution
--------
Increased the argument, called value, only in the end of increaseByOne.
Previously it was increased both at the beginning and the end of the
function.
```

References:

- A note about Git commit messages:
  http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

Pull Requests

The Pull Request message guideline in bullet form:

- **The title**
    - Same as Trello card/ Github Issue.
- The message body has up to 5 sections, each separated by an empty line
    - **Description**
        - At least two sentences.
    - **Overview**
        - **Story**
            - Start with “Story”.
            - Link text is the name of the Github story issue.
            - Link URL is the URL to the Github story issue.
        - **Affected Pull Requests**
            - Starts with “Affected Pull Requests”.
            - Comma separated list of all known other Pull Requests that in any way risk colliding with this one.
                - Link text is the name of the Pull Request.
                - Link URL is the URL to the Pull Request.
        - **Affected Critical Components**
            - Title “Affected Critical Components” in **bold.**
            - List with one bullet for each affected Pull Request.
                - Starts with the component name followed by colon.
                - Description
                    - What changes has been made to the component.
                    - At least two sentences.
        - **Affected Pull Requests**
            - Title “Affected Pull Requests” in **bold.**
            - List with one bullet for each affected Pull Request.
                - Start with a linked title followed by colon
                    - Link text is the name of the Pull Request.
                    - Link URL is the URL to the Pull Request.
                - Description
                    - How the Pull Requests affecting each other.
                    - At least two sentences.
        - **Pictures** (If the UI has been changed)

Example:

The document indexes in the database are all evenly numbered.

(Github PR title)

```
Resolves the issue regarding all documents having even numbered indexes in the
database. The counter function is now increasing the index by one instead of by
two.

 - **Github Issue:** [The document indexes in the database are all evenly
numbered](https://github.com/input-output-hk/sales-frontend/issues/1234)
 - **Critical Components:** None
 - **Affected Pull Requests:** [Add edit counter to all documents in the
database](https://github.com/input-output-hk/sales-frontend/pull/1357)
```

Release Notes

The Pull Request message guideline in bullet form:

- **Title**
  - Same as the release name.
- **Feature section** (if any features has been implemented)
  - Title; Features.
  - List with one bullet for each feature Pull Request **as a Link.**
    - Link text is the description of the Pull Request.
    - Link URL is the URL to the Pull Request.
- Bug fix section (if any bug fixes has been implemented)
  - Title; Bug fixes.
  - List with one bullet for each bug fix Pull Request **as a link.**
    - Link text is the description of the Pull Request.
    - Link URL is the URL to the Pull Request.

Example:

```
## Release - X

Features
 * [A super cool feature to always finish the sprints on time. The solution let the user chose when the feature was finished if a flux capacitor is connected to the application.](https://github.com/input-output-hk/sales-frontend/pull/2015)

Bug fixes
 * [Resolves the issue regarding all documents having even numbered indexes in the database. The counter function is now increasing the index by one instead of by two.](https://github.com/input-output-hk/sales-frontend/pull/1357)
```

Testing

Testing allows you to ensure your application works the way you think it does, especially as your codebase changes over time. If you have good tests, you can refactor and rewrite code with confidence. Tests are also the most concrete form of documentation of expected behavior, since other developers can figure out how to use your code by reading the tests.

Automated testing is critical because it allows you to run a far greater set of tests than you could do manually, allowing you to catch regression errors immediately.

References:

- Meteor testing: https://guide.meteor.com/testing.html

Setting Up Your System

You have to install these two npm packages globally to be able to run our tests:

- npm i -g chimp
- npm i -g spacejam

All tests of the sales application live in the attainenroll/test directory are split up according to to their type.

Entire books have been written on the subject of testing, so we will simply touch on some basics of testing here. The most important thing to consider when writing a test is what part of the application you are trying to test, and how you are verifying the behavior works.

Unit Test

If you are testing one small module of your application, you are writing a unit test. You'll need to stub and mock other modules that your module usually leverages in order to isolate each test. You'll typically also need to spy on actions that the module takes to verify that they occur. Unit tests are run by Mocha.js

Unit test mode:

- Doesn't eagerly load any of our application code as Meteor normally would.
- Does eagerly load any file in our application (including in imports/folders) that look like *.test[s]* or *.spec[s]*

References:

- Mocha.js: http://mochajs.org/

Integration

If you are testing that multiple modules behave properly in concert, you are writing an integration test. Such tests are much more complex and may require running code both on the client and on the server to verify that communication across that divide is working as expected.

Typically an integration test will still isolate a part of the entire application and directly verify result in code. Meteor offers a “full application”, or integration test mode.

Integration test mode:

- It loads test file matching .app-test[s] and .app-spec[s]...
- It does eagerly load our application code as Meteor normally would.
- Sets the Meteor.isAppTest flag to be true (instead of the Meteor.isTest flag).

This means that the entirety of your application (including for instance the web server and client side router) is loaded and will run as normal. This enables you to write much more complex integration tests and also additional files for acceptance tests.

End-to-End

If you want to write a test that can be run against any running version of you application and verifies at browser level that the right things happen when you push the right buttons, then you are writing an acceptance test.

Such tests typically try to hook into application as little as possible, beyond perhaps setting up the right data to run a test against.

We use Chimp.js to write our acceptance tests in Cucumber scenarios with great Meteor integration. Chimp.js integrates everything you need and is based on Webdriver.io for browser automation.

Running the Tests

To run the tests, youhave to run the `./bin/run/dev` or use a command to start up the Mongo database `mongod --port 3100` before you run the acceptance tests. This is because the test application used by Chimp.js actually hooks up a new chimp_db with the local Meteor server from the dev application. You can inspect that in Robomongo with the same connection like for the dev application.

Running

First you have to tag the features and/or scenarios you want to run with a `@watch` tag like this:

```
@watch # If you put it here, all scenarios in the feature will be run
Feature: Enroll

  @watch # if you put it here then only this scenario is run (you can tag multiple
escenarios like this)
  Scenario Outline: A new user enrolls correctly
    Given There is a <ROLE> enroll link created
      And I go to enroll screen
    When I finish completing residence selection
      And I finish completing contact information for <ROLE>
      And I finish selecting agreements for <ROLE>
      And I finish selecting document type
    And I skip upload documents
    Then I should see a thank you message for enrolling

    Examples:
      | ROLE              |
      | buyer             |
      | distributor       |
```

Then execute ./bin/test/end-to-end-watch and it will only pick up the ones tagged with @watch.

References:

- Robomongo: https://robomongo.org/

Running Tests Locally

To manually execute the tests on your local machine, you should execute any of the following commands from the terminal:

- bin/test/all-run Runs end-to-end, integration and unit tests at once.
- bin/test/end-to-end-run - Runs every Chimp test,
- bin/test/end-to-end-watch - Runs every Chimp test tagged with @dev or @watch.
- bin/test/integration-run - Runs every Mocha test inside '/test/integration' folder.
- bin/test/integration-watch - Runs every Mocha test inside '/test/integration' on the browser
- bin/test/unit-run - Runs every Mocha test inside inside '/test/unit folder
- bin/test/unit-watch - Runs every Mocha test inside inside '/test/unit folder on the browser
- bin/test/packages-run - Tests calculateOrderFulfillmentAmount method with latests packages.

Running Tests on CircleCI

CircleCI is a Continuous Integration platform, which will automatically execute 'unit-run', 'integration-run' and 'end-to-end-run' scripts every time there's a change (i.e. a commit) in the remote repository (located in GitHub).

However, if you intend to re-run these tests, you are able to select one execution from the list and click 'Rebuild'.

Sometimes, there are some tests that fail due to timeout issues. In that case, it's strongly recommended to rebuild the test execution choosing the option 'Rebuild without cache'.

It is also possible to integrate CircleCI with Github, in order to block PRs from merging if tests are failing in Circle. (This should only be set if we're 100% sure that tests are not failing randomly due to timeout issues or false positives).

Regarding to configuration, CircleCI uses node version 4.5.0, and updates to the latest chrome driver version. The entire configuration for CircleCI can be found in the file 'circle.yml'.

References:

- CircleCI: https://circleci.com/

API

As stated before, the Sales application interacts with other external systems. For this purpose there is an API (Application Programming Interface) that allows them to communicate by simple REST style HTTP requests.

There are two basic API services; to interact with the backend, that handles all bitcoin network events and the JSON Generator, which exposes necessary Vending Machine methods.

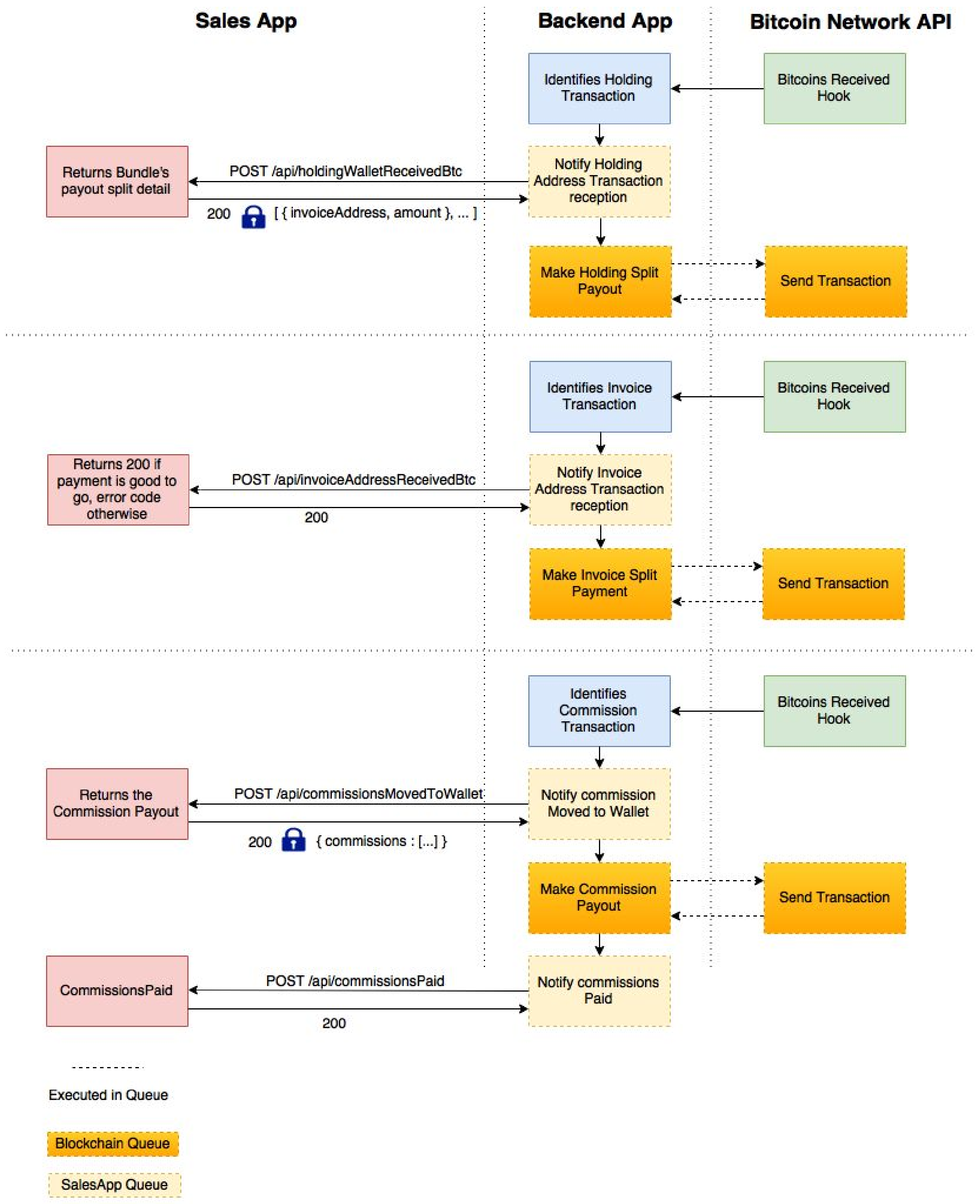
Backend

Backend application is the Sales application interface with the Bitcoin network. It’s a wrapper that simplifies all payments and notifications made over that network in a way that abstract the sales process of the actual payments and transactions. It also serves as a security layer as it gives a degree of independence to the business logic from the Wallets and private keys associated to the process.

Backend application is documented in detail in other documents. This section focuses on the API services that the Sales application uses to receive Backend related events. Notice that even though these endpoints were designed for Backend application interaction, they are actually public and demand no authentication so that they could be manually triggered easily if necessary.

There is a prearranged protocol for endpoints’ HTTP Status codes response, on top of the basis standard (200 OK, 4XX/5XX error) we distinguish between two main sets to indicate whether the Backend needs to retry again later with this notification or if it should stop. Any error code bigger than 555, will bring an end to this notification, otherwise the Backend will retry if there is an error. Besides that, any expected business error will have a descriptive text message as well.

Every sensitive data return by these API Endpoints, will be encrypted with a predefined key arranged with the Backend; this guarantees both authorization and data integrity.


Sales App
Backend App
Bitcoin Network API
Identifies Holding Transaction
Bitcoins Received Hook
Returns Bundle's payout split detail
POST /api/holdingWalletReceivedBtc
Notify Holding Address Transaction reception
200
[{ invoiceAddress, amount }, ...]
Make Holding Split Payout
Send Transaction
Identifies Invoice Transaction
Bitcoins Received Hook
Returns 200 if payment is good to go, error code otherwise
POST /api/invoiceAddressReceivedBtc
Notify Invoice Address Transaction reception
200
Make Invoice Split Payment
Send Transaction
Identifies Commission Transaction
Bitcoins Received Hook
Returns the Commission Payout
POST /api/commissionsMovedToWallet
Notify commission Moved to Wallet
200
{ commissions : [...] }
Make Commission Payout
Send Transaction
CommissionsPaid
POST /api/commissionsPaid
Notify commissions Paid
200
Executed in Queue
Blockchain Queue
SalesApp Queue

Backend notifies Sales application on Bitcoin Network events.

Endpoints Details:

- Holding Wallets Received Bitcoins Notification

Holding Wallet is where Bank Bundled payments are received. One single address in that Wallet is linked to a particular Bundle and funds are also sent by one single transaction. This endpoint is where the Sales application gets notified that one of those transactions has been confirmed. Sales application will search for the corresponding bundle by the received address, verify the amount and return the list of individual orders (invoice address) that belong to that bundle with each corresponding amount. This list is returned as a json encrypted response payload.

```
POST /api/holdingWalletReceivedBtc body
      {
            "holdingWalletAddress":"address",
            "satoshisReceived":"#ofsatoshisreceived",
            "transactionId": "transactionId"
      }
→ 200 signed(*) body
   { "payout" : [ { invoiceAddress": "invoiceAddress", "amount":"amount"}, ... ] }
→ < 555 => (any error < 555 triggers retry)
→ > 555 => Stop Call
```

(*) Note: The payload body must be signed with the backend given key.

- Invoice Address Received Bitcoin

Each Invoice Ticket is associated with an individual address, this address belongs to HD Multisig Wallets linked as well as with the responsible distributor of this sale. Whenever a payment is received and confirmed, Sales application will receive a notification on this endpoint.

The notification will specify the amount of confirmations received, and the sales application will respond to basically 2 states, zero confirmation (aka lightning signal) and any confirmation. First one will just record the event with the transaction Id and the amount, but when confirmed, the payment is verified and emails are sent. On a successful response to this last event, Backend will split the funds to the exodus address and the Commission address.

```
POST /api/invoiceAddressReceivedBtc body
{
      "invoiceAddress": [String],
      "satoshisReceived": [Integer],
      "transactionId": [String],
      "date" : [DateTime],
      "confirmations" :  [Integer],
}  → { 200 | 4xx | 5xx}
```

- Commission are ready to be paid

Once an Invoice has been paid and the transaction is confirmed and approved by the invoiceAddressReceivedBtc endpoint, commissions must be paid as well to each corresponding distributor. This endpoint notifies the Sales application that the Backend is ready to make those payments, and expects the detailed information of which address and quantities should be done. For this, the sales application will respond - if successful - with an encrypted payload containing the distributor's personal address and the commission share amount (in satoshis) corresponding to each one. The sum of all, should add up to the commission address totals.

```
POST /api/commissionsMovedToWallet body
      {"invoiceAddress": "invoice-address",
      "commissionWalletAddress":"commission-address",
      "satoshisAmount": "satoshis-amount",
      "transactionId": "transactionId",
      "date": "tx-date"}
→ 200 signed(*) body
      {"commissions": [{"commissionAddresss": "address", "amount": "#ofsatoshis"}]
→ < 555 => (any error < 555 triggers retry)
→ > 555 => Stop Call
```

(*) Note: The payload body must be signed with the backend given key.

- Commissions Paid

Once the commission payout transaction is confirmed, this endpoint should be called to finish the payment process.

```
POST /api/commissionsPaid body
      {"invoiceAddress": "invoice-address", "transactionId": "transactionId"}
→ 200 => Success/Confirmation
→ < 555 => (any error < 555 triggers retry)
→ > 555 => Stop Call
```

JSON Generator

The JSON Generator is the intermediate data export step between the Sales application and the AVVM. The actual data is read directly from the same database as the Sales application but updates have to be done through API calls.

Set Invoice Ticket PII Hash

Saves a PII hash to the database together with an invitation email. The request should be sent as a `POST` package to `/api/setInvoicePiiHash`.

Headers

Type	Description
`authorization`	API authentication token. Example value: `Bu4vH1vW-i1PVmmXg-peRFLqVH-hZU5xZtO`

Body

The body should be formatted as a JSON string with the following fields.

Field name	Description
`ticketId`	The id of the ticket for which the PII hash should be updated. The id format depends on the id incrementer used when inserting documents in the database. Example id: `2EBPJaDa8mGfGcxfR`
`piiHash`	The PII hash to be set. The hash is a SHA256 hash of a concatenation of specific database values. Example hash: `5uOHZGlIH4XBACULMXsdz12ON89/9sYmlqAgjrSMNFM=`
`email`	The invitation email used by the JSON Generator. The email format has to comply with the W3C recommendations. Example email: `email@domain.com`
`comment` (optional)	An optional comment to store with the PII hash. Example comment: `A comment about the PII hash generation.`

Response Codes

Status code	Content	Description
`200`		Successfully updated the `piiHash`.
`400`	Malformed request body	The `ticketId`, `piiHash`, `email` keys or the body itself is missing.
`400`	Invalid piiHash format	The `piiHash` key was found but the value wasn't a string.
`400`	Invalid email format	The `email` key was found but the value didn't match the recommended email format by W3C.
`400`	Invalid comment format	The `comment` key was found but the value wasn't a string.
`400`	Invoice Ticket not found for ticketId: <ticketId>	The `ticketId` key was found but the value didn't match any existing ticket id.
`400`	Invoice Ticket <ticketId> is not ready to set piiHash yet	The `ticketId` key was found but there is no Ada passcode for the ticket.
`400`	Invalid body content, couldn't save the data	The content couldn't be saved to the database.
`401`	Token Authorization error	No authentication token was provided or an incorrect token was used.

Example: Request and response packages

```
POST /api/setInvoiceTicketPiiHash HTTP/1.1
host: test.domain.com
authorization: Bu4vH1vW-i1PVmmXg-peRFLqVH-hZU5xZtO

{
  "ticketId": "2EBPJaDa8mGfGcxfR",
  "piiHash": "5uOHZGlIH4XBACULMXsdz120N89/9sYmlqAgjrSMNFM=",
  "email": "email.domain.com",
  "comment": "This email is probably faulty."
}
```

Package with faulty email address (no @ sign)

```
HTTP/1.1 400 Bad Request
content-length: 20
content-type: text/plain; charset=utf-8
date: Thu, 22 Dec 2016 05:57:22 GMT
status: 400

Invalid email format
```

Response with notification about the faulty email format

Advanced Features

Jobs and Workers

Many state transitions and background processes are run by Jobs and Workers. These workers are integrated into the Sales application using *vsivsi:job-collection* library. This library is a powerful and easy to use job manager for Meteor applications, it allows:

- Scheduled jobs to run (and repeat) in the future, persisting across server restarts.
- Create repeating jobs with complex schedules using [Later.js].
- Move work out of Meteor's single threaded event-loop.
- Enable work on computationally expensive jobs to run anywhere, on any number of machines.
- Track jobs and their progress, and automatically retry failed jobs.
- Easily build an admin UI to manage all of the above using Meteor's reactivity and UI goodness.

In Sales-app we use endless workers that run along the application's life, each worker will be identify by it's jobName (works as key). On startup, *init.js* will start each of the registered jobs.

```
const job = new Job(Jobs, jobName, {})
.retry({ retries: Jobs.forever, wait: callInterval })
.repeat({ repeats: Jobs.forever, wait: callInterval })
.save();
```

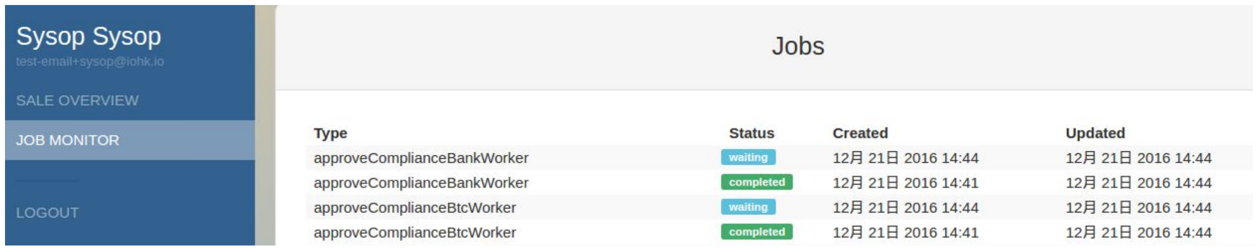
Every worker will need to

- Declare an unique name
- Mongodb cursor to pick the data it will act on
- Task (function) it wants to execute over each item returned by that cursor.

For example, the worker that sends funds received confirmation looks like:

```
Jobs.registerWorker({
 jobName: 'sendConfirmFundsReceivedEmailWorker',
 cursor: function getFundsReceivedConfirmedTickets() {
   return InvoiceTickets.find({ state: 'fundsReceivedConfirmed' });
 },
 task(ticket) {
   const options = { ticketId: ticket._id };
   sendET205ConfirmFundsReceived(options);
   updateInvoiceTicket('sendConfirmFundsReceived', ticket);
 }
});
```

Jobs are recorded to the database on the “jobs.jobs” collection, but a simplified version can be access by the sysop role in the application, the dashboard displays current job status:



Type	Status	Created	Updated
approveComplianceBankWorker	waiting	12月 21日 2016 14:44	12月 21日 2016 14:44
approveComplianceBankWorker	completed	12月 21日 2016 14:41	12月 21日 2016 14:44
approveComplianceBtcWorker	waiting	12月 21日 2016 14:44	12月 21日 2016 14:44
approveComplianceBtcWorker	completed	12月 21日 2016 14:41	12月 21日 2016 14:44

References:

- Later.js: https://bunkat.github.io/later/
- Vsivsi:job-collection: https://atmospherejs.com/vsivsi/job-collection

Ada Reservation

ADA reservation process is composed by a set of controlled sales, it can be generally turned on or off by the flag “salesStarted” and allowed in particular for each region by "residenceCountriesAllowedSale".

Each period (tranche) has a predefined total of USD equivalent of ADA ("totalAmountAvaible") available to be exchanged, this amount can be exceeded by a certain percentage ("overCapacityTolerance").

All of this is configured in the application settings file and needs to be set before each deploy:

```
// file path: "/config/settings.json"
{
 "public": {
   "salesStarted": true,
   "residenceCountriesAllowedSale": ["KR","JP"],
   "salesLimits": {
     "currentTranche": "t3",
     "tranches": {
       "t1": {},
       "t2": {},
       "t3": {
         "totalAmountAvailable": 14000000,
         "overCapacityTolerance": 0.2
       },
       "t4": {}
     }
   },
   ...
  }
}
```

Sales Predictor

The module in charge of checking if the sales goal has been reached and if a ticket can be sold, is called "SalesPredictor".

To do so, it's instanced with the following parameters:

```
// file path: '/imports/lib/sales-predictor.js'
export default class SalesPredictor {
 /*
  numbers:
  totalAmountAvailable:    Total amount available in this tranche
  totalAmountAvailableCap: Total amount available + over cap
  overCapacityTolerance:   Overcapacity tolerance

  functions:
  amountInvoicedOrPaid:  Amount of invoiced or received payment in this tranche
  amountPaid:   Amount of received payment in this tranche
  */
 constructor(options) {
   this.totalAmountAvailable = parseInt(options.totalAmountAvailable, 10) || 0;

   this.overCapacityTolerance = parseFloat(options.overCapacityTolerance) || 0;

   this.amountInvoicedOrPaid =
     options.amountInvoicedOrPaid || function() { return 0; };

   this.amountPaid  = options.amountPaid  || function() { return 0; };

   this.overCapacityToleranceValue =
     this.totalAmountAvailable * this.overCapacityTolerance;

   this.totalAmountAvailableCap =
     this.totalAmountAvailable + this.overCapacityToleranceValue;
 }
...
}
```

"amountPaid" is calculated as the sum of all ticket's amounts in paid states (*) for current tranche. And "amountInvoicedOrPaid", will add all tickets in invoiced states (**) on top of that.

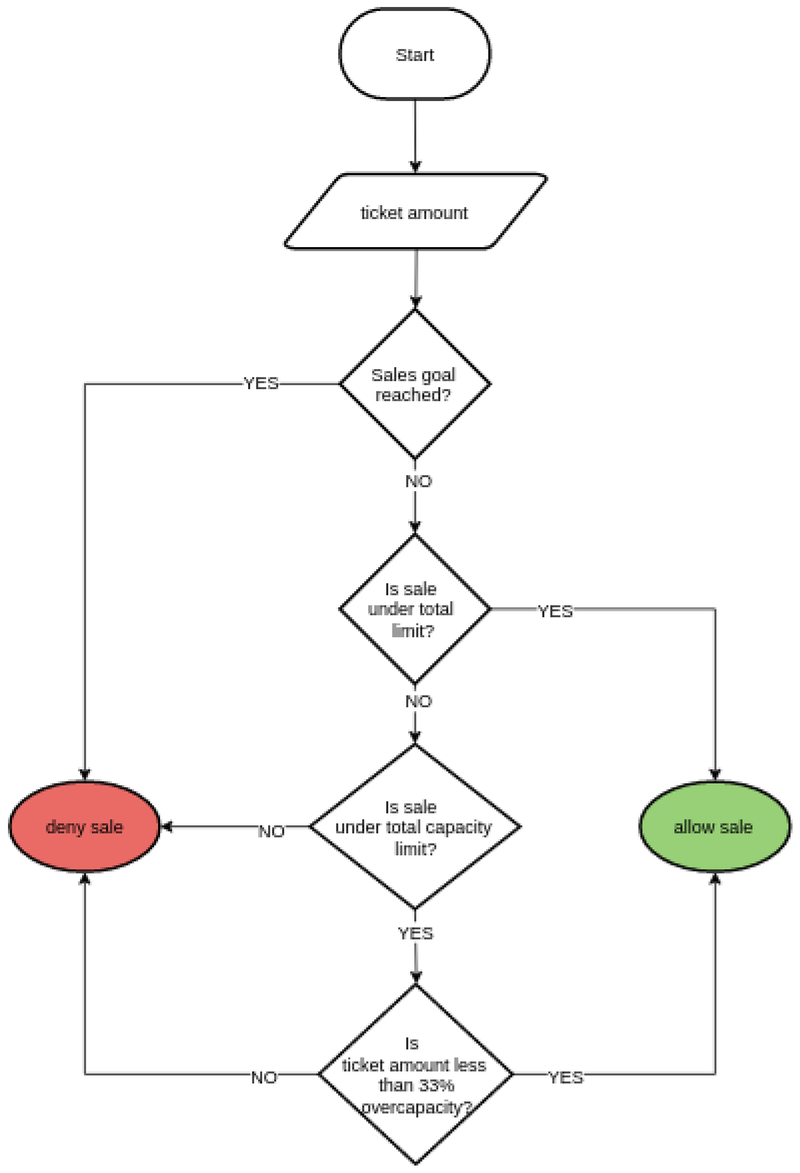
(*) Paid states:

- satoshisReceived
- adaPasscodeAssigned
- receiptSent

(**) Invoiced states:

- saleStartedBtc
- saleStartedBank
- complianceApprovedBtc
- complianceApprovedBank
- fundsReceivedConfirmed
- fundsReceivedConfirmedSent
- ticketPrepared
- ticketBundled
- ticketFinalized
- invoiceSentBank
- fundsReceived
- invoiceSentBtc
- invalidTicketApproved
- invalidFundsReceived.

A new ticket it can be sold only if the salesPredictor allows it. The validation method is called “allowSale()” and follows the next flow chart:


Start
ticket amount
Sales goal reached?
YES
NO
Is sale under total limit?
YES
NO
Is sale under total capacity limit?
NO
YES
deny sale
allow sale
Is ticket amount less than 33% overcapacity?
NO
YES

Invoice Ticket State Machine

In order to control the invoice tickets flow, we use a meteor plugin that exposes a *state-machine* to manage all the different states and the transitions between them.

Depending on the case, the transitions on the machine could be triggered by a worker or an user action.

These transitions are defined according to the business logic, and the invoice state machine will only allow a ticket to change its state if the state exists and the transition is valid.

Example: of usage in a worker (from server side)

```
'use strict';


import { Jobs } from '/lib/collections/jobs.js';
import { sendInvoiceEmail } from '../../imports/server/email-actions.js';
import { updateInvoiceTicket } from '/imports/server/invoice-ticket-state-machine';


Jobs.registerWorker({
 jobName: 'sendInvoiceWorker',
 cursor: function getTickets() {
   return InvoiceTickets.find({ state: {$in:
       ['complianceApprovedBank', 'complianceApprovedBtc']
   }});
 },
 task: function sendInvoice(ticket) {
   const options = { ticketId: ticket._id };
   const changeToState = 'sendInvoice' + ticket.paymentOption;
   sendInvoiceEmail(options);
   updateInvoiceTicket(changeToState, ticket);
 },
});
```

The stateMachine is defined and created in file “invoice-ticket-state-machine”. In that file you will find the exposed method “updateInvoiceTicket”, that manages the state transitions, validations and data updates for the ticket.

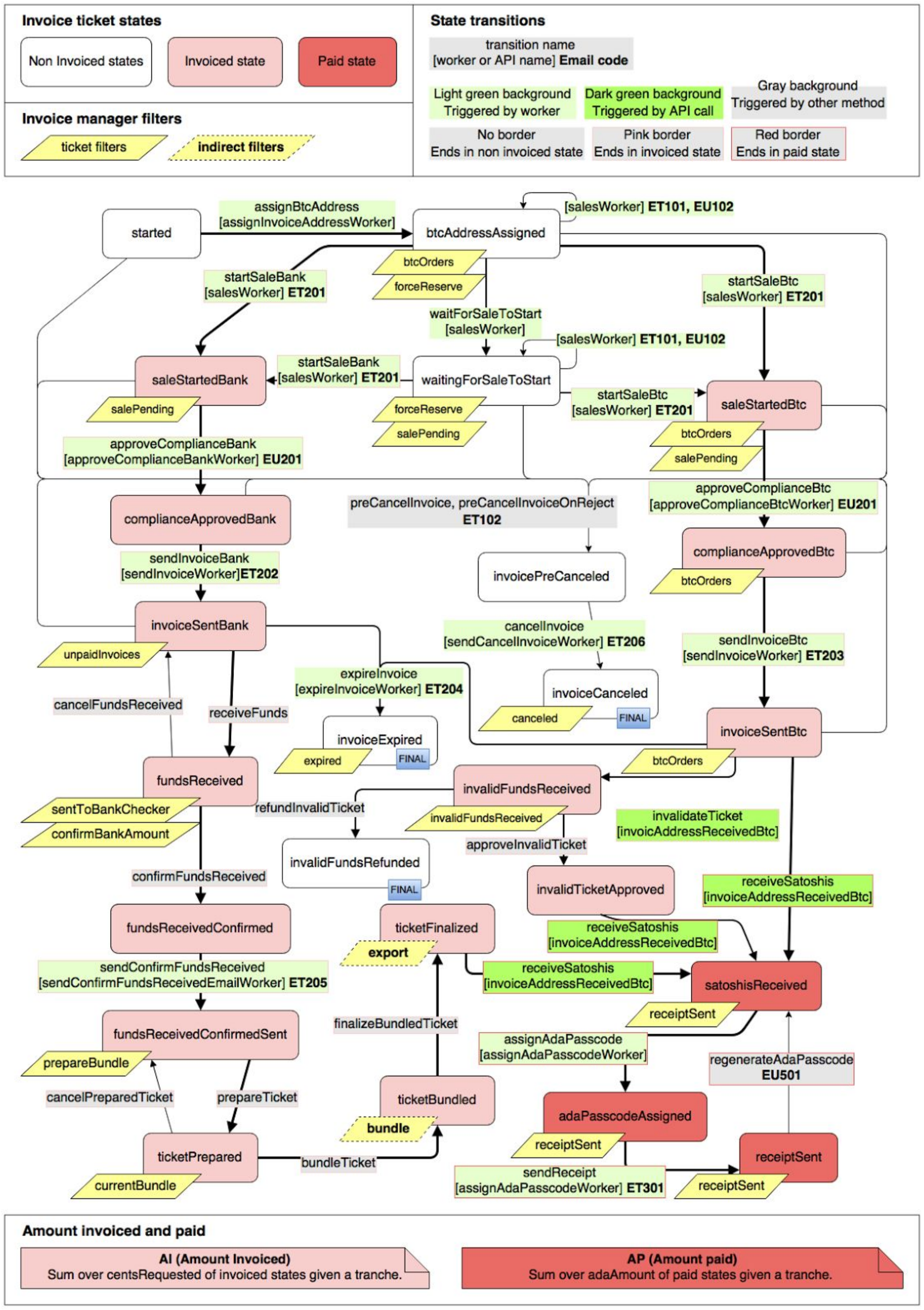
Example: of a state transition triggered by an user action (from client side)

```
// in file: '/client/views/user-overview/partials/invoice-ticket-action-cancel.js'
// user action:
events: {
 'click button.cancel-ticket'() {
   confirmModal(
       i18n('modals.cancelTicket'),
       ()=> {
         this.data.invoice.changeState(
           'preCancelInvoice',
           undefined,
           (err) => {i18nAlert(err, I18N_ERROR_NAMESPACE);}
         );
       }
   );
 }
}
...

// in file: '/lib/collections/invoice-tickets.js'
// invoice method:
changeState: function(event, modifier, handlerError, handlerSuccess) {
 Meteor.call('changeState', event, this, modifier, function(err) {
   if (err && _.isFunction(handlerError)) {
     handlerError(err);
   } else if (_.isFunction(handlerSuccess)) {
     handlerSuccess();
   }
 });
},
```

The correct way to use the state-machine from client side is using the method "changeState" of the invoice. This method internally uses the meteor call “changeState” that checks user role and creates a record of the action (in the collection stateTransitionRecords) before it uses the “updateInvoiceTicket” state-machine method.

Full diagram of the Invoice state machine:



References:

- State Machine package: https://github.com/jakesgordon/javascript-state-machine#usage

Refcodes

Refcodes are authentication codes that allow external users to interact with their data. This means either adding new data or changing current data. A refcode can be bound to an email-address or userId and has no predefined structure.

Refcodes are generated by using a md5 hash from a random number. Refcodes expire 24 hours after they are created except for 4 special types.

Router Validation

The refcodes are validated in the router, and then they are marked as isActive = false in the database, so they cannot be used again (Unless the refcode timetype is 'unlimited' which is a possible case for enroll refcodes). Then the application will redirect and call a certain template depending on the ref type, which is defined in the router.

The application has the following refcode types:

Type	Description	Expires
emailVerification	This verifies a user's email	Y
confirmAddress	This confirms the wallet address update	Y
confirmEmailAccount	This confirms an email-account	Y
confirmEmailChange	This confirms the change in email-address	Y
reset	This allows a user to reset the password	Y
re-generate-all	This allows the user to regenerate all of the ADA codes	Y
re-generate-one	This allows the user to regenerate one of the ADA codes	Y
signup	This allows a user to create a password on the first login as a distributor	N
buyer	Refcode for a buyer to enroll	N
distributor	Refcode for a distributor to enroll	N
partner	Refcode for a partner to enroll	N

Refcode documents can contain the following keys:

Type	Description
_id	Document id
clicked	Shows how many times a refcode has been used
reftype	The type of refcode
refcode	The code that is used for verification of the ref
emailto	The email linked to the refcode
timetype	This specifies how many times a refcode can be used
originatorId	This is the userId of the owner of the refcode, i.e. the distributor that supplied it
createdAt	The date the refcode was created
isActive	Whether the refcode is active
name	Name of the person bound to the refcode
notes	Any notes left on the refcode
userId	The id of the user bound to the code
emailAddress	The email-address that corresponds to the refcode
invoiceTicketId	The invoice ticket bound to the refcode

Example: Refcode documents

Type	Description
emailVerification	{ "_id" : "LMtzpcYwSYggkvbDY", "clicked" : 0, "reftype" : "emailVerification", "refcode" : "5db3aa940d0b0a386f16", "emailto" : "test-email+xxx@iohk.io", "timetype" : "onetime", "originatorId" : null, "createdAt" : ISODate("2016-12-20T05:56:52.559Z"), "isActive" : true }
confirmAddress	{ "_id" : "B5dpQPa78anW67qJQ", "clicked" : 0, "reftype" : "confirmAddress", "refcode" : "r39cd78dccf7bb7c9a12d", "timetype" : null, "name" : null, "notes" : null, "userId" : "22ThpRGZtdKxRBKFp", "address" : "1FRqs8e8sCY8AQTdCoALJLJDNNqdDxCYVk", "originatorId" : null, "createdAt" : ISODate("2016-12-20T06:41:59.912Z"), "isActive" : true }
confirmEmailAccount	{ "_id" : "PLp4hzarKaJaRF8W3", "clicked" : 0, "reftype" : "confirmEmailAccount", "refcode" : "r4656d0ca8e1266cf0abb", "timetype" : null, "name" : null, "notes" : null, "userId" : "22CZvqFHAgHQhRWuf", "emailAddress" : "test-email+xxx@iohk.io", "originatorId" : null, "createdAt" : ISODate("2016-12-20T06:50:19.718Z"), "isActive" : true }
confirmEmailChange	{ "_id" : "Nw5kpFo5apyvB4gmp", "clicked" : 0, "reftype" : "confirmEmailChange",

	"refcode" : "r80707e208adb504734c4", "timetype" : null, "name" : null, "notes" : null, "userId" : "22CZvqFHAgHQhRWuf", "emailAddress" : "test-email+xxx@iohk.io", "originatorId" : null, "createdAt" : ISODate("2016-12-20T06:51:34.370Z"), "isActive" : true }

reset	{
	"_id" : "22ziMACE7jtHKWqxC",
	"reftype" : "reset",
	"refcode" : "s0eb666896e9eb9j3r3m7as026292dba7",
	"emailto" : "test-email+xxx@iohk.io",
	"originatorId" : null,
	"createdAt" : ISODate("2016-07-16T06:28:39.005Z"),
	"isActive" : false,
	"clicked" : 0
	}

re-generate-all	{
	"_id" : "ghuapwfAbRPB42JZj",
	"clicked" : 0,
	"reftype" : "re-generate-all",
	"refcode" : "ra6927f4f850602a9f23e",
	"timetype" : null,
	"name" : null,
	"notes" : null,
	"userId" : "NNTPyahg4AqG7hBZG",
	"emailto" : null,
	"originatorId" : null,
	"createdAt" : ISODate("2016-12-20T07:16:08.806Z"),
	"isActive" : true
	}

re-generate-one	{ "_id" : "KCPTjrwAhvFovSdZD", "clicked" : 0, "reftype" : "re-generate-one", "refcode" : "rd7b8a4c7cb7f8ecf3c25", "timetype" : null, "name" : null, "notes" : null, "userId" : "NNTPyahg4AqG7hBZG", "emailto" : null, "invoiceTicketId" : "2aAyvaMc2FcFPG6Xg", "originatorId" : null, "createdAt" : ISODate("2016-12-20T07:15:20.352Z"), "isActive" : true

	}
signup	{ "_id" : "229ZREjWBw6qPJDBK", "reftype" : "signup", "refcode" : "s0eb666896e9eb9j3r3m7as026292dba7", "emailto" : "test-email+xxx@iohk.io", "originatorId" : null, "createdAt" : ISODate("2016-08-24T08:03:31.899Z"), "isActive" : false, "clicked" : 0 }
buyer	{ "_id" : "22CEgSYgDirKshtfN", "reftype" : "buyer", "name" : "Trum", "timetype" : "onetime", "isActive" : false, "originatorId" : "EzpBZ6RDDTxraZHNG", "refcode" : "s0eb666896e9eb9j3r3m7as026292dba7", "clicked" : 0, "notes" : "Dope", "createdAt" : ISODate("2016-04-21T04:24:38.829Z") }
distributor	{ "_id" : "22DuPnsxybzSDvFh8", "createdAt" : ISODate("2015-09-07T09:32:09.235Z"), "refcode" : "s0eb666896e9eb9j3r3m7as026292dba7", "clicked" : 0, "originatorId" : "dmkd8fGiMjEKwXwYQ", "notes" : "Dope", "name" : "Trum", "timetype" : "unlimited", "distlevel" : 2, "reftype" : "distributor", "isActive" : false }
partner	{ "_id" : "2BhmGfz6xwbZkbudT", "distlevel" : 0, "createdAt" : ISODate("2015-09-29T02:30:13.512Z"), "notes" : "Dope", "timetype" : "onetime", "name" : "Trum", "isActive" : false, "refcode" : "s0eb666896e9eb9j3r3m7as026292dba7",

	"clicked" : 0, "reftype" : "partner", "originatorId" : "xDMihrT8EzqfxY6uv" }

Languages, Standards and Frameworks

Standards

JSON

JSON (JavaScript Object Notation) is an open-standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs.

In the application all the records are stored into JSON documents (see section Databases - MongoDB) and also is used as a data format for the communication between the Sales application and the Backend.

Example: JSON in the application

```
// ref document from the refs collection.
{
  "_id" : "22N5msmBzyvByaoti",
  "clicked" : 0,
  "reftype" : "signup",
  "refcode" : "s1f588932417f1384703251cd65059e39",
  "emailto" : "sample-user@iohk.io",
  "originatorId" : null,
  "createdAt" : ISODate("2016-02-20T08:19:24.562Z"),
  "isActive" : true
}
```

HTTP

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

Sales application uses HTTP to serve all it’s frontend content, although it follows modern web browsers standards, we recommend Google Chrome (Version 54+).

i18n

i18n (Internationalization) is a system for allowing the application to support multiple languages. Most important part when using i18n system, is to replace all strings containing text with i18n objects.
We use following meteor packages to implement i18n:

- Anti:i18n
- Gwendall: simple-schema-i18n
- Gwendall: auto-form-i18n

Currently all 18n files are stored at sales-frontend/attainenroll/lib. Format for the files is as follows:

i18n_viewinks.js

```
i18n.map('en', {
  viewlinks: {
    partner: 'Partner Links',
    distributor: 'Distributor Links',
    buyer: 'Buyer Links',
  }
});

i18n.map('ja', {
  viewlinks: {
    partner: 'パートナー用紹介URL',
    distributor: '代理店用紹介URL',
    buyer: '交換希望者用紹介URL'
  }
});

i18n.map('ko', {
  viewlinks: {
    partner: '파트너용 소개 URL',
    distributor: '대리점용 소개 URL',
    buyer: '교환 희망자용 소개 URL',
  }
});

i18n.map('zh', {
  viewlinks: {
    partner: '合作夥伴介绍连结',
    distributor: '代理店介绍连结',
    buyer: '希望兑换者介绍连结',
  }
});
```

The application currently supports up to four languages(depending on the part of the application), so the i18n file is split in four i18n maps. In each map there is a property containing a list of translations for that language.

Property name is usually the same as the file name. To insert the translated i18n string in html code, use the following syntax:

```
{{ i18n 'viewlinks.buyer' }}
```

To access the i18n string in javascript use:

```
i18n('viewlinks.buyer');
```

To switch between languages, pass in language code(‘en’, ‘ja’, ‘ko’, ‘zh’) as parameter to this function:

```
i18n.setLanguage('ja');
```

Missing translations are marked with !!! and the language code for ease of searching:

```
i18n.map('en', {
  viewlinks: {
    partner: 'Partner Links',
  }
});

i18n.map('ja', {
  viewlinks: {
    partner: '!!!ja Partner Links',
  }
});

i18n.map('ko', {
  viewlinks: {
    partner: '!!!ko Partner Links',
  }
});

i18n.map('zh', {
  viewlinks: {
    partner: '!!!zh Partner Links',
  }
});
```

Unneeded translations are marked with !!!- and the language code:

```
i18n.map('ko', {
  viewlinks: {
    partner: '!!!-ko Partner Links',
  }
});
```

References:

- I18n package: https://atmospherejs.com/anti/i18n
- Anti:i18n: https://github.com/anticoders/meteor-i18n
- Gwendall: simple-schema-i18n: https://github.com/gwendall/meteor-simple-schema-i18n
- Gwendall: auto-form-i18n: https://github.com/gwendall/meteor-autoform-i18n

DDP

Distributed Data Protocol (or DDP) is a client-server protocol created for use by the Meteor framework.

It used to power the querying and updating a server-side database and for synchronizing such updates among clients. In order to handle It, DDP uses the publish-subscribe messaging pattern.

Example:

```
// On the Server: '/server/publications.js'

Meteor.publish('jobs', function() {
 if (Roles.userIsInRole(this.userId, ['sysop'])) {
   return Jobs.find({}, secureJobsOptionsByRole({}, 'sysop'));
 }
 return null;
});
```

```
// On the Client: '/client/views/jobs-monitor/jobs-monitor.js'

TemplateController('jobsMonitor', {
 onCreated() {
   Meteor.subscribe('jobs');
 },
 helpers: {
   jobs() {
     return Jobs.find({}, {sort: {type: 1, created: -1, updated: -1}});
   }
 },
// file continues..
});
```

With this, the implementation of DDP in Meteor handles the rest; It intelligently polls the database to pick up changes on the jobs collection and pushes them down to the client. DDP can also simulate the model code on the client side, so the screen is updated immediately without waiting for the network response.

References:

- DDP: https://github.com/meteor/meteor/blob/master/packages/ddp/DDP.md#ddp-specification

Languages

ECMAScript 6

Ecmascript 6 is the latest (written at 21 December 2016) version of the JavaScript standard. The application uses many features that come with this version.

Example:

```
import { markJobAsReady } from '/imports/server/workers';
import { preCancelInvoicesOnReject } from '/server/invoice-manager';
const checkRevieweeExistance = (userId, callerMethod) => {
 const reviewee = Meteor.users.findOne(userId);
 if (reviewee === undefined || reviewee === null) {
   Logger.error('[RPC][Warning] ', callerMethod, ': User not found.', 'userId:',
userId, 'customerServiceId:', this.userId);
   throw new Meteor.Error('userNotFound');
 }
 return reviewee;
};
```

References:

- ES6: http://es6-features.org

HTML

HTML is the main markup language used in the project. Every template used, uses this markup.

Example:

```
<template name="buyers">
<div id="buyers-screen" class="data-screen">
   <header>
       <div class="headline">
           <h1>{{i18n "buyers.headline"}}</h1>
           {{#if showAddBuyerButton}}
               <button class="add-buyer">{{i18n "buyers.addBuyer"}}</button>
           {{/if}}
       </div>
       <p class="buyers-count">
         {{i18n "buyers.count.youHave"}}
         {{buyersCount}}
         {{i18n "buyers.count.buyers"}}
       </p>
       <a href="#" class="toggle-all">
           {{#if areAllOpen}}
               {{i18n "buyers.closeAll"}}
           {{else}}
               {{i18n "buyers.openAll"}}
           {{/if}}
       </a>
   </header>
   <div class="buyers-list">
       {{#each buyer in currentUser.buyers}}
           <div class="row">
               {{> buyerPanel buyer=buyer isOpen=areAllOpen }}
           </div>
```

```
        {{/each}}
    </div>
    {{#if hasMoreToLoad}}
        <div class="load-more-results">{{> spinner}}</div>
    {{/if}}
</div>
</template>
```

LESS

Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that is more maintainable, themable and extendable.

LESS files are the main form of styling used for the templates in the code base.

Example: LESS

```
.account-info-panel {
 .panel-heading {
   padding-bottom: 0;
   background-color: white;
   h1 {
     margin-bottom: 1em;
   }
 }
```

Example: Template using the styles from LESS

```
<template name="accountInfoPanel">
<div class="account-info-panel panel panel-default">
 <div class="panel-heading">
   <h1>{{ i18n 'accountInfo.account.title' }}</h1>
   <!-- Nav tabs -->
   <ul class="nav tabs" role="tablist">
     <li role="presentation" class="active">
       <a href="#information" aria-controls="information" role="tab">
         {{ i18n 'accountInfo.tabs.information' }}
       </a>
     </li>
     <li role="presentation">
       <a href="#agreements" aria-controls="agreements" role="tab">
         {{ i18n 'accountInfo.tabs.agreements' }}
       </a>
     </li>
   </ul>
 </div>
```

References:

- LESS: http://lesscss.org/#

Cucumber

Cucumber is a tool for running automated tests written in plain language, allowing it to be easily read and understood by anyone on your team.

This tool understands and uses Gherkin to write test scenarios in a more 'human' language, and is used within Chimp.js.

References:

- Cucumber: https://cucumber.io/

Shell Script

A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter. The various dialects of shell scripts are considered to be scripting languages. Many Sales application script are shell script commands, as they are intended to be run at SO level. For example there are shell scripts to run the application, tests, and to deploy.

Example: Sales application run shell script ./bin/run/dev

```
#!/usr/bin/env bash
echo "Sourcing ./bin/env/development.sh as environment ..."
source ./bin/env/development.sh
cd attainenroll/
echo "Running meteor with --settings ../config/develop.json ..."
meteor "$@" --settings ../config/develop.json --no-release-check
```

Perl

Perl is a family of high-level, general-purpose, interpreted, dynamic programming languages.

The Perl languages borrow features from other programming languages including C, shell script (sh), AWK, and sed. They provide powerful text processing facilities without the arbitrary data-length limits of many contemporary Unix command line tools, facilitating easy manipulation of text files. It is commonly used by Web programmers to create scripts for Web servers.

Perl commands are used internally in some of Sales application bash scripts. For example, in deploy script, we can found:

```
# Collaps settings JSON data one level
echo -ne "$WAIT Preparing settings and secrets..."
perl -i -pe 'BEGIN{undef $/;} s/\s*\{(.*)\}\s*/\1/smg' $TEMP_SETTINGS
```

References:

- Perl: http://perldoc.perl.org/index.html

PowerShell

Powershell is a console in Windows which can execute .ps1 scripts and works similarly, this is similar to Shell Scripts in unix/linux based systems.

Example: PowerShell script that starts the Meteor server

```
echo "Sourcing ./bin/env/development.ps1 as environment ..."
. ./bin/env/development.ps1
cd ./attainenroll
echo "Running meteor with --settings ../config/development/settings.json ..."
meteor --settings ../config/development/settings.json
```

Frameworks

Meteor

Meteor is a full-stack JavaScript platform for developing modern web and mobile applications. It integrates with MongoDB and uses the Distributed Data Protocol (DDP) and a publish–subscribe pattern to automatically propagate data changes to clients without requiring the developer to write any synchronization code.

Meteor uses **data on the wire**, meaning the server sends data, not HTML, and the client renders it. It also provides **full stack reactivity**, allowing the UI to seamlessly reflect the true state of the world with minimal development effort.

- Meteor version: 1.4.2.3
- Node version: 4.6.0
- npm version: 3.10.8

Example: Publish-subscribe code

```
// Server
Meteor.publish('reviewRecords', function() {
 if (Roles.userIsInRole(this.userId, Meteor.users.getComplianceRoles())) {
   return ReviewRecords.find({});
 }
 return undefined;
});

// Client
Meteor.subscribe('reviewRecords');
```

Example: collection code

```
// To create a new collection
ReviewRecords = new Mongo.Collection('reviewRecords');

// Find, update, insert
ReviewRecords.find();
ReviewRecords.insert(reviewRecord);
ReviewRecords.update({_id: reviewRecordId}, { $set: {date: new Date()}});
```

References:

- Meteor: https://www.meteor.com/

Blaze

Blaze is the Meteor's default frontend rendering system.

- Its templates consists of views and controllers
  - Views are written in **Spacebars** which is handlebars-like templating language with reactively changing datacontext.
  - Controllers are written in **Javascript**.
- It uses Tracker to automatically keep track of when to recalculate each template. Because of this, developers don't have to explicitly declare when to update the DOM, or even perform any explicit "data-binding."

We use Blaze to create sales application user interface. We also use template-controller which provides a very thin layer of syntactic sugar on top of the standard API.

References:

- Blaze: http://blazejs.org/index.html
- Tracker: https://docs.meteor.com/api/tracker.html
- Template-Controller: https://github.com/meteor-space/template-controller

Example:

Investigator-tabs.html

```
<template name="investigatorTabs">
  <div class="row">
    <div class="col-md-3">
      {{> counterButton label=(i18n 'investigator.tabs.underInvestigation')
                         value="underInvestigation"
                         count=getCountValue
                         isActive=isSelected
                         onClicked=onTabSelected
                         buttonId='underInvestigation' }}
    </div>
    <div class="col-md-3">
      {{> counterButton label=(i18n 'investigator.tabs.all')
                         value="all"
                         count=getCountValue
                         isActive=isSelected
```

```
                                   onClicked=onTabSelected
                                   buttonId='all' }}
      </div>
    </div>
</template>
```

Investigator-tabs.js

```
TemplateController('investigatorTabs', {
  props: new SimpleSchema({
    selected: {
      type: String
    },
    onTabChanged: {
      type: Function
    },
    counts: {
      type: Object,
      blackbox: true
    }
  }),
  helpers: {
    isSelected: function() {
      return (tab) => this.props.selected === tab;
    },
    onTabSelected: function() {
      return (tabClicked) => this.props.onTabChanged(tabClicked);
    },
    getCountValue() {
      return (tab) => this.props.counts[tab];
    }
  }
});
```

Node

Node.js is a server-side JavaScript runtime environment based on Google's V8 JavaScript engine. Meteor.js is supported by Node.js and its' packages. We are using Node.js on its own in very few places in application, mostly connected with tests.

Example: Cache build and dependencies for CircleCI

```
#!/usr/bin/env node

const path = require('path');
const spawn = require('child_process').spawn;
const baseDir = path.resolve(__dirname, '../..');
const srcDir = path.resolve(baseDir, 'attainenroll');

const cacheMeteor = function() {
  console.log('Caching build & dependencies (can take a while the first time)');
  const childProcess = spawn('meteor', ['--raw-logs', '--settings',
'../config/testing.json'], {
    cwd: srcDir,
    env: process.env
  });
  childProcess.stdout.setEncoding('utf8');
  childProcess.stderr.setEncoding('utf8');
  childProcess.stdout.on('data', function(line) {
    process.stdout.write(line);
  });
  childProcess.stderr.on('data', function(line) {
    process.stderr.write(line);
  });
  const exitAfterBuild = function exitAfterBuild(line) {
    if (line.indexOf('App running at') !== -1) {
      childProcess.kill();
      console.log('Done caching build & dependencies');
    } else if (
      line.indexOf('Your application is crashing') !== -1 ||
      line.indexOf('Errors prevented startup') !== -1) {
      childProcess.kill();
      console.error('There were issues whilst trying to cache build &
dependencies');
      throw new Error(line);
    }
  };
  childProcess.stdout.on('data', exitAfterBuild);
  childProcess.stderr.on('data', exitAfterBuild);
};

cacheMeteor();
```

References:

- Node.js: https://nodejs.org/en/

Mocha.js

Mocha.js is a JavaScript test framework running on Node.js and in browser. We are using it for unit and integrations tests.

Example: Testing if invoice ticket schema validation is working correctly.

```
describe('Invoice Tickets Schema Validations', function() {

  it('ticket with piiHash is valid', function() {
    const obj = { adaVendingInvitation: { piiHash: 'n8fh3hfeHGfrv89hfuihv8HUIb43h'
}};
    const isValid =
Schemas.InvoiceTicket.namedContext("piiHash_validation").validateOne(obj,
"adaVendingInvitation");
    expect(isValid).isTrue;
  });

  it('piiHash gets removed after Schemma clean', function() {
    const obj = { buyerId: '1',  adaVendingInvitation: { piiHash:
'n8fh3hfeHGfrv89hfuihv8HUIb43h'} };
    const validated = Schemas.InvoiceTicket.clean(obj);
    expect(validated.buyerId).equals('1');
    expect(validated.adaVendingInvitation).to.be.undefined;
  });

});
```

References:

- Mocha.js: https://mochajs.org/

Chai.js

Chai is an assertion library for node and the browser. We are using it in combination with Mocha.js.

Example:

```
expect(validated.buyerId).equals('1');
expect(validated.adaVendingInvitation).to.be.undefined;
```

References:

- Chai: http://chaijs.com/

Chimp.js

Chimp.js is a test framework with good support for meteor and CI(continuous integration). It supports Mocha, Jasmine and Cucumber for writing the tests. In this project we are using Cucumber to write ours. We are using Chimp.js only for end-to-end tests.

Example: Evaluate if the text in “reviewer-name” is empty. If empty, test will pass.

```
import { waitAndGetText } from '/test/end-to-end/tests/_support/webdriver';

module.exports = function() {
  this.Then(/^the reviewer name is empty$/, function() {
    const name = waitAndGetText(".reviewer-name");
    expect(name).to.eq('');
  });
};
```

References:

- Chimp.js: https://chimp.readme.io/

Databases

MongoDB

Meteor.js comes with MongoDB as the default database. MongoDB is a NoSQL open-source database, and it's using JSON like objects for schemas.

Example: MongoDB insertion

```
Commissions.insert({
      distributorId: commission.distributorId,
      ticketId: invoiceTicket._id,
      invoiceNumber: invoiceTicket.invoiceNumber,
      buyerId: invoiceTicket.buyerId,
      originator: originator,
      payoutAmount: commission.payoutAmount,
      payoutTransactionId: invoiceTicket.payoutTransactionId,
      paidAt: new Date(),
      createdAt: invoiceTicket.satoshisReceivedAt
});
```

References:

- MongoDB: https://docs.mongodb.com/

Improvement Suggestions

Test Suite Improvement

Because sales application has lots of functionality, testing thoroughly takes a long time. Especially end-to-end tests since running all scenarios takes about 30 minutes. Part of this is because some steps are repeatedly used (for example When I login) which takes substantial amount of time. Because of this, it is important to write a test code that would not fail or else someone who is not responsible for the task may have to fix the problem which wastes their working time.

It is hard to write tests that involves animations, because test suite ignores the animation and when trying to do the next step - it fails, because the contents that the animation should show are not yet displayed. Major example of this is modals. When modals pop up, you have to do an action, such as confirm or cancel. After waiting for the modal to disappear, you continue with the next action. Our tests occasionally fails here, because they don't wait for modal to disappear. We have tried to solve this with various ways and spend a lot of time, but it is still occurring sometimes.

CircleCI greatly helps us reduce our testing time but it is not as fast as running the tests on our own machines, which results in longer tests.

We have discussed about switching over to other packages but some were outdated or has negative impact on our testing environment. It would help us a lot if we can find a better testing suite for improving our test environment for future.

Middleman Attack Yielding

As you have seen, there is a two way communication going on between the sales application and backend-app in order to fulfill the payments and reflect the status on a user level. Given Galaxy environment characteristics, there is no way to set a point-to-point close network between those two applications, what would be ideal, and as a result of this flaw (Galaxy has dynamic IPs) both applications have open endpoints.

Although this was secured on Backend side by adding Token authentication and response encryption, sales application endpoints are completely open. This means that even if every payment is secure and there is no way to manipulate actual money, a middleman attack can mess around with the sales application quite bad, manipulating tickets payments status. Although that would require a deep understanding of its functionality and data.

A possible solution is to add authentication on sales application as well, to verify that any request on the Backend API is from a trusted third party.

Toggle Sorting Button for Invoice Managers

In the exporter, the HIM bundle list the tickets are sorted newest at the bottom, but sometimes it's useful to switch the sorting order to find newest tickets.

Add a sorting button that toggles ascending/descending sorting order to the invoice managers, to allow the invoice manager to choose how to sort the tickets/bundles.

Block Password Reset Emails for X Amount of Time

Currently "Send password reset email" button can be clicked infinite times, and infinite amount of emails will be sent. If a malicious person wants, he would be able to saturate the email service.

Reset email should be sent only when a certain time has been passed before the the first email was sent (e.g. 30 mins).

Turn Pay Page into an Invoice Ticket Status Page

Convert the pay page into a ticket state page. The page will show the tickets current and previous stages.

The key point is to show when the ticket

- is created (date, amount)
- has Ada reserved (date)
- need payment (date, amount, address)
- is expired (date)
- is paid (date, amount)
- has passcode assigned (date)

This page is read only.

Show The Buyer's Residence Country on the Invoice Ticket Card in the Invoice Queue

Currently in the invoice queue the card component for invoices doesn't show the buyer's residence country. It would be better if the invoice manager could see that field.

Add Backend Status Info to Sysop Dashboard

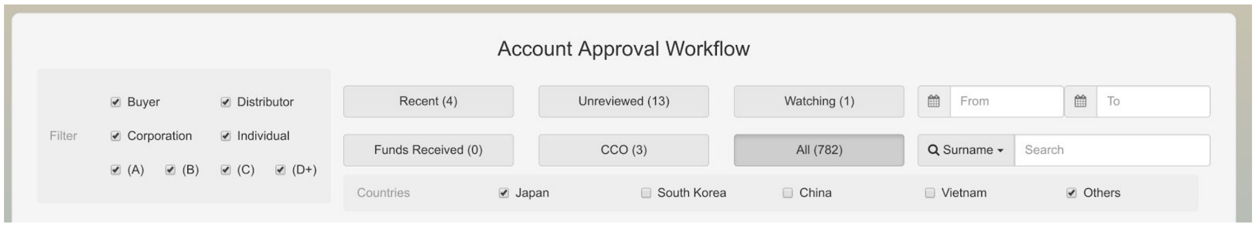
Currently some back-end status parameters are not visible in the sysop dashboard, they need to be queried specifically to detect any abnormal situation. A quick overview dashboard could be valuable to have an instant info about a potential issue.

Some useful information could be:

- Health-check data (backend version for example)
- Sales-app notifications queue amount (also details perhaps?)
- White-list queue amount (also details perhaps?)
- Transaction queue amount (also details perhaps?)
- Holding Wallet balance
- Commission Wallet balance
- Search tool by invoice wallet or address

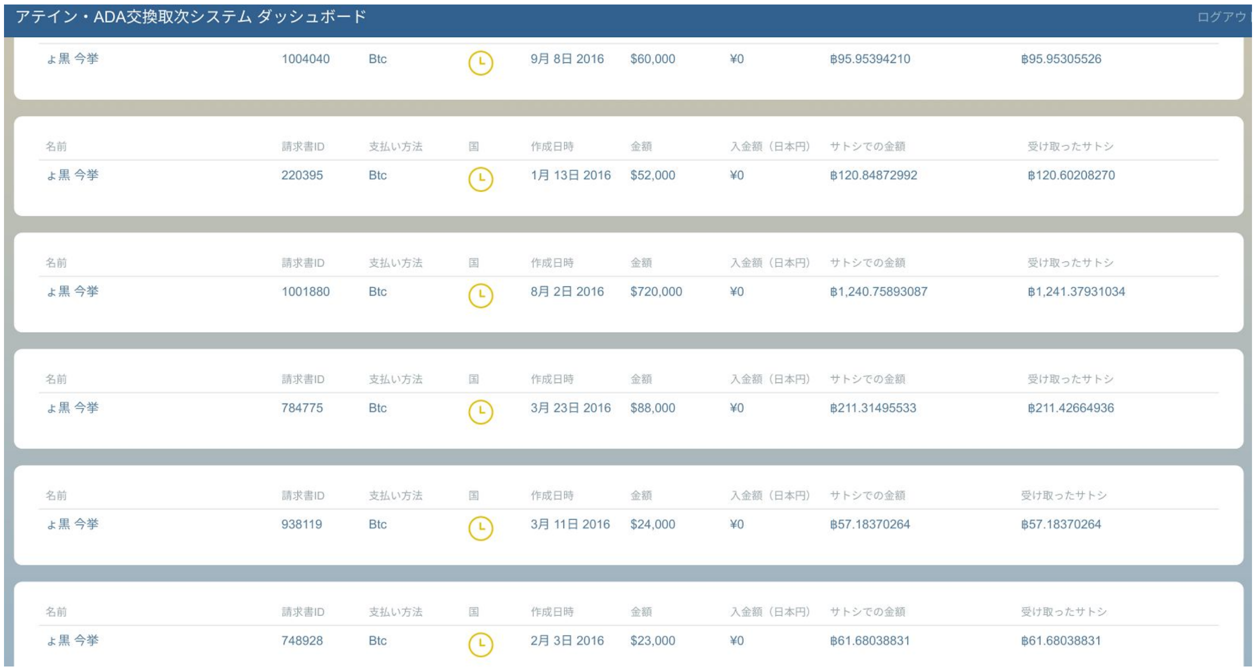
Extend the Region Filter for Compliance Officers

The number of countries where sales are available has increased, thereby we should extend the region filter to include China (and maybe also Vietnam) into the list.



Sort the Invoice List in Review User Page

The list of invoice tickets in the user review page is seemingly unsorted. When a D+/D+ user is reviewed this can be very confusing for the compliance officer. A more reasonable listing would be a descending sorting by creation date (or similar).

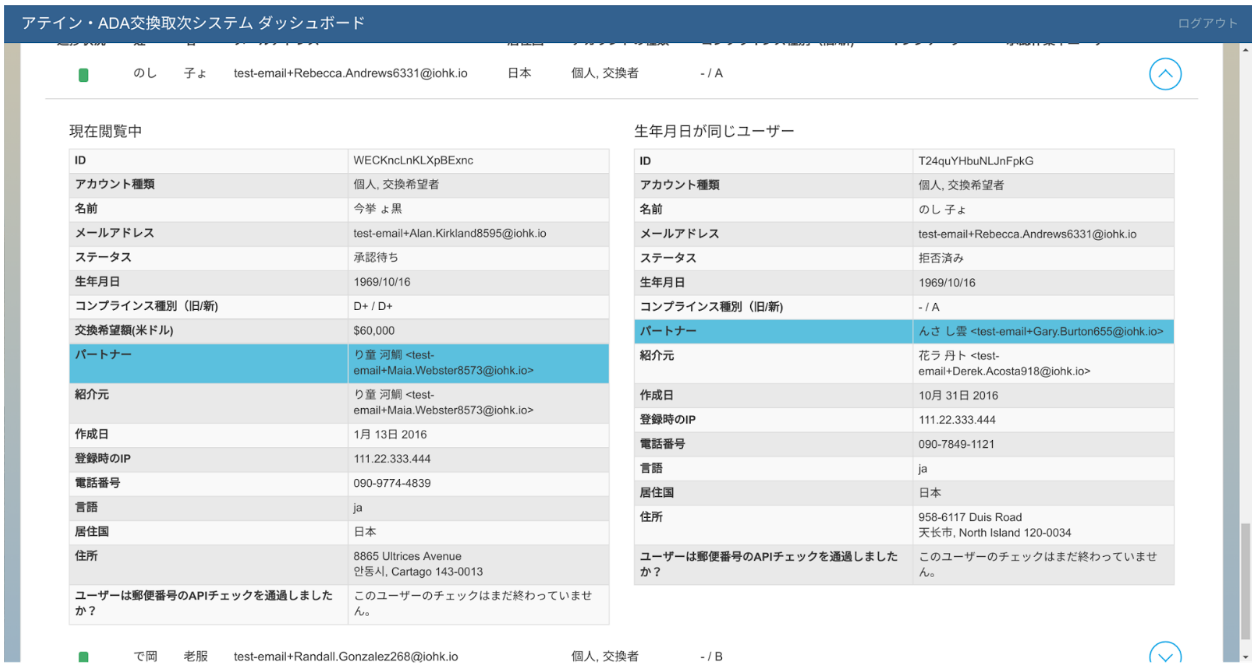

アテイン・ADA交換取次システム ダッシュボード

ログアウ

名前	請求書ID	支払い方法	国	作成日時	金額	入金額（日本円）	サトシでの金額	受け取ったサトシ
よ黒 今挙	1004040	Btc	🕒	9月 8日 2016	$60,000	¥0	฿95.95394210	฿95.95305526
よ黒 今挙	220395	Btc	🕒	1月 13日 2016	$52,000	¥0	฿120.84872992	฿120.60208270
よ黒 今挙	1001880	Btc	🕒	8月 2日 2016	$720,000	¥0	฿1,240.75893087	฿1,241.37931034
よ黒 今挙	784775	Btc	🕒	3月 23日 2016	$88,000	¥0	฿211.31495533	฿211.42664936
よ黒 今挙	938119	Btc	🕒	3月 11日 2016	$24,000	¥0	฿57.18370264	฿57.18370264
よ黒 今挙	748928	Btc	🕒	2月 3日 2016	$23,000	¥0	฿61.68038831	฿61.68038831

Hover in Birth Date Comparison Table Doesn't Line Up

The user listings under birth date comparison in review user page has a comparison table for the two users. When hovering a field it highlights the field in blue color for both users. But if one of the users has a field that the other lacks, then the comparison doesn't line up in the UI.



🟩 のし 子ょ test-email+Rebecca.Andrews6331@iohk.io 日本 個人, 交換者 - / A

現在閲覧中

ID	WECKncLnKLXpBExnc
アカウント種類	個人, 交換希望者
名前	今挙 よ黒
メールアドレス	test-email+Alan.Kirkland8595@iohk.io
ステータス	承認待ち
生年月日	1969/10/16
コンプライアンス種別（旧/新)	D+ / D+
交換希望額(米ドル)	$60,000
パートナー	り童 河鯛 <test-email+Maia.Webster8573@iohk.io>
紹介元	り童 河鯛 <test-email+Maia.Webster8573@iohk.io>
作成日	1月 13日 2016
登録時のIP	111.22.333.444
電話番号	090-9774-4839
言語	ja
居住国	日本
住所	8865 Ultrices Avenue 안동시, Cartago 143-0013
ユーザーは郵便番号のAPIチェックを通過しましたか？	このユーザーのチェックはまだ終わっていません。

生年月日が同じユーザー

ID	T24quYHbuNLJnFpkG
アカウント種類	個人, 交換希望者
名前	のし 子ょ
メールアドレス	test-email+Rebecca.Andrews6331@iohk.io
ステータス	拒否済み
生年月日	1969/10/16
コンプライアンス種別（旧/新)	- / A
パートナー	んさ し雲 <test-email+Gary.Burton655@iohk.io>
紹介元	花ラ 丹ト <test-email+Derek.Acosta918@iohk.io>
作成日	10月 31日 2016
登録時のIP	111.22.333.444
電話番号	090-7849-1121
言語	ja
居住国	日本
住所	958-6117 Duis Road 天长市, North Island 120-0034
ユーザーは郵便番号のAPIチェックを通過しましたか？	このユーザーのチェックはまだ終わっていません。

🟩 で岡 老服 test-email+Randall.Gonzalez268@iohk.io 個人, 交換者 - / B

Frontend
Technical Documentation

Better Ticket Information in User Summary View

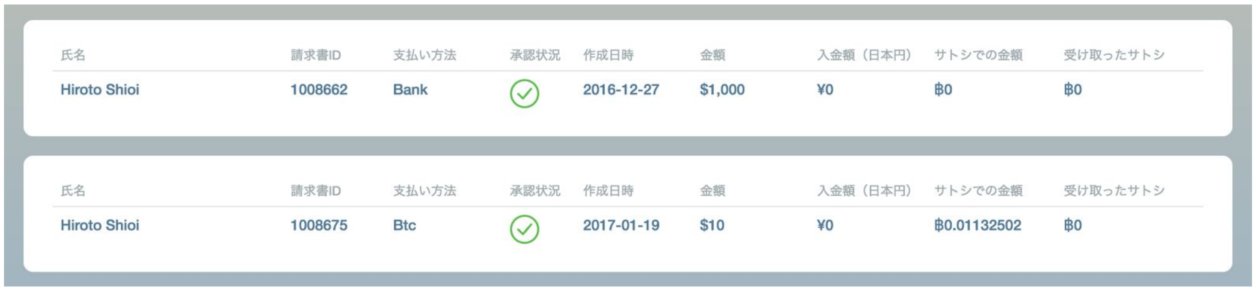
Some information on the enlisted tickets are unnecessary and some are missing for the purpose of being shown in the user summary view.

Unnecessary:

- Name
- State (user status)

Missing:

- Unreserved, Reserved, Paid, Canceled
- State (invoice ticket)



氏名	請求書ID	支払い方法	承認状況	作成日時	金額	入金額（日本円）	サトシでの金額	受け取ったサトシ
Hiroto Shioi	1008662	Bank	✓	2016-12-27	$1,000	¥0	฿0	฿0

氏名	請求書ID	支払い方法	承認状況	作成日時	金額	入金額（日本円）	サトシでの金額	受け取ったサトシ
Hiroto Shioi	1008675	Btc	✓	2017-01-19	$10	¥0	฿0.01132502	฿0

Special Holiday Settings

More than once manual work has been done to prevent ticket expiration due to holidays. These tasks are time consuming and prone to errors.

Add an array field `public.expireTickets.nonExpirationHolidays` to the settings file preventing ticket expiration for the given days. The program should handle the listed days in the same way, regarding ticket expiration, as it currently handles weekends when `public.expireTickets.X.afterInvoiceSent.businessDays` is set to `true`.

```
    "expireTickets": {
      "bank": {
        "whilePendingCompliace": {
          "days": 5,
          "businessDays": true
        }
        "afterInvoiceSent": {
          "days": 3,
          "businessDays": true
        }
      },
      "btc": {
        "whilePendingCompliace": {
          "days": 5,
          "businessDays": false
        }
        "afterInvoiceSent": {
          "days": 3,
          "businessDays": false
        }
      }
    },
```

Add Expiration for Users Pending Compliance

This feature was already planned to be implemented some time ago, but has yet to be done.

Add an field `public.expireTickets.X.whilePendingCompliace` similar to `public.expireTickets.X.afterInvoiceSent` to the settings file, to expire tickets in the `states saleStartedBank` and `saleStartedBtc`. The program should handle the expiration in the same way as for `public.expireTickets.X.afterInvoiceSent`.

```
    "expireTickets": {
      "bank": {
        "whilePendingCompliace": {
          "days": 5,
          "businessDays": true
        }
        "afterInvoiceSent": {
          "days": 3,
          "businessDays": true
        }
      },
      "btc": {
        "whilePendingCompliace": {
          "days": 5,
          "businessDays": false
        }
        "afterInvoiceSent": {
          "days": 3,
          "businessDays": false
        }
      }
    },
```

Known Bugs

Zip Api Check not Always Saving During Enrolment

During enrolment, when a user fills in a zip code that the google api doesn't recognize, a checkbox shows up that forces a user to acknowledge the zip code.
This however, seems to fail to save to the user document sometimes for an unknown reason.

When the Pricing Providers are Offline, then the Application Will Set satoshisExpected, centsAskPrice and btcUsd to 0

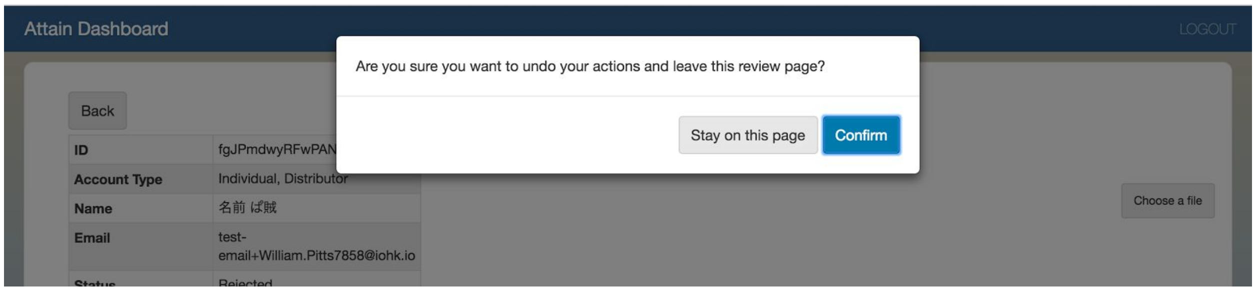
This has its ups and downs, it can be good as it prevents purchases using outdated rate, but there is currently no way to for the frontend-backend interaction to solve a situation where a user pays a ticket while the pricing providers are down.

When Reviewing a User, and CO Rotates the Image, the Next Image Viewed Will Also Be Rotated

When you login as a compliance officer and rotate a picture in review-user view, all other pictures will be display with the same rotation when clicked.

When You Login as an Investigator and Press “Back” After Reviewing, Misleading Message Appears

As an Investigator, when you enter to review a user and then press Back, the message says:
“Are you sure you want to undo your actions and leave this review page?” which could be misleading.



Are you sure you want to undo your actions and leave this review page?

ID	fgJPmdwyRFwPAN
Account Type	Individual, Distributor
Name	名前 ぱ賊
Email	test-email+William.Pitts7858@iohk.io
Status	Rejected

When Commissions are Made, the Origin’s Email are Stored in the Document and Remains Unchanged Even If User Change Their Email Address

When user pays the ticket and commissions are made, origin’s email are stored in the commission document. This means that even if user changes their email address, commission’s origin will remain unchanged. This may cause confusion to the distributors.

During Logout or When Switching Views, Console Errors Can Appear

When logging out as an user, error messages appear on the console log. However, this does not affect the application’s performance.

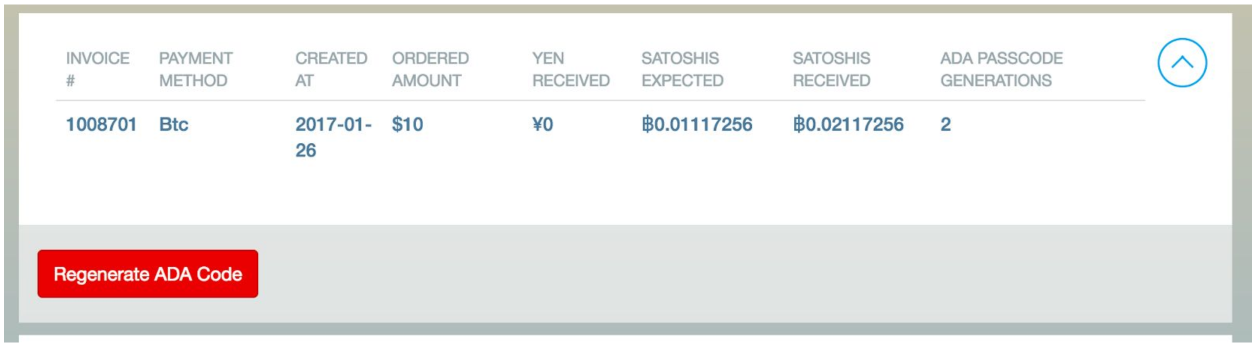
The Payment Information on the Email and Payment Link for Bank Tickets are Inconsistent

Here is the current format of the payment link for bank tickets.
Email
Payment amount: Requested amount + Attain fee.
Payment Link
Payment amount: Requested amount.

Even if the payment link isn't sent out any more, the conflicting information can be confusing if someone enters it.

'RegenerateAdaPasscode' Button is Still Visible Even Though the Ticket is in the State Where Passcode Cannot Be Regenerated

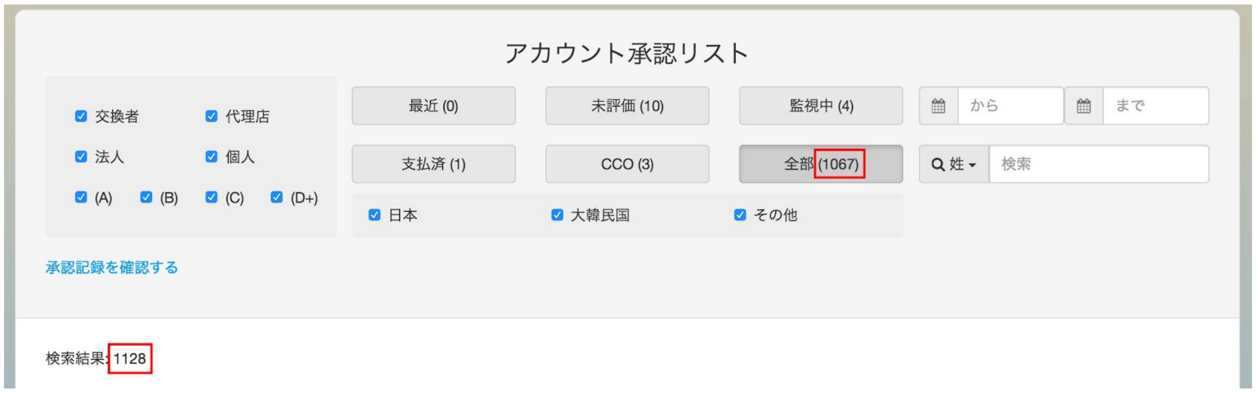
User can see the 'Regenerate ADA Code' button on the tickets where the state is not 'receiptSent' and cannot regenerate passcodes. They can try to regenerate them but they'll get an error 'Please specify a message' on the console log.



INVOICE #	PAYMENT METHOD	CREATED AT	ORDERED AMOUNT	YEN RECEIVED	SATOSHIS EXPECTED	SATOSHIS RECEIVED	ADA PASSCODE GENERATIONS
1008701	Btc	2017-01-26	$10	¥0	฿0.01117256	฿0.02117256	2

Counter On the Tab and Search Result Counter Show Different Number in Compliance View

When you login as a Compliance officer, the counter on the tab and search result show different numbers. This is because the counter on the tab is not counting the users in which their residenceCountry is neither "Japan" or "Korea".



アカウント承認リスト

✓ 交換者 ✓ 代理店
✓ 法人 ✓ 個人
✓ (A) ✓ (B) ✓ (C) ✓ (D+)

✓ 日本 ✓ 大韓民国 ✓ その他

検索結果: 1128

User Can Enter String on Phone Field on Enrollment

Buyer and Distributor can enter String on phone number field in enrollment. This means that user can enter emails, sentences etc. and still get their personal information valid on enrollment.