Story UFC-372: Import Rules

The following document will go over the design of adding import rules to the design system, as outlined in Story ticket UFC-372.

Acceptance Criteria

For reference, the following are the acceptance criteria for this ticket:

- I want import rules that can map descriptions to accounts.
- I want import rules that can map placeholder (i.e. raw value) accounts to accounts.
- I want import rules that can rewrite raw descriptions to a given description.
- I want import rules that can map descriptions/accounts to a type.
- I want import rules to support regexes and constant values.
- I want import rules that can have multiple conditions.
- I want import rules that can have multiple actions.
- I want to CRUD my import rules.
- I want to see which rules have been applied during the Adjust Transactions step.
- I want to be able to toggle applying/not applying rules during the Adjust Transactions step.
- I want to be able to add rules during the Adjust Transactions step.
- I want the rules system to be built with being used externally in mind (e.g. as part of an API client).

Out of Scope

- Automatically creating rules based on user behaviour.
- Manually prioritizing rules for tie-breaking.

Design Brainstorming

rules system is definitely something we can take inspiration from.

Conflicting Rules

How do we resolve import rules that have matching conditions? That is, if two rules have conditions that both apply to a single transaction, how do we tie-break? I guess the most naive way would be to apply them in oldest-to-newest (creation/update) order.

But has an explicit 'priority' property for manually ordering rules... I wonder if our users would want that... in the name of simplicity, we shouldn't support it out of the gate.

Although, has the more powerful automatic ranking system, wherein rules with more specific conditions are applied last. I guess that means that we'd need to determine this ranking ahead of time so that we can run them in that order. And then I guess last-updated would be the tie breaker if two rules ranked the same.

Rules applying to Rules

What should we do if a rule modifies e.g. the description and then another rule depends on the description?

I guess, nothing. We just apply the rules in the (ranked) order and let things play out.

Using Workers

I definitely think that applying rules to transactions (especially a *lot* of transactions) would benefit from the parallel processing of using web workers.

Will Workers even Work?

I don't even know if workers will work properly — or optimally. My main concern is the fact that the code we'd be splitting up references a *lot* of other code (i.e. basically all the models), so I wonder if the code splitting process for the worker script will require that all of that code is brought in.

Will have to test to find out.

Might end up just putting a loading spinner on the third step to at least improve the UX somewhat.

Decision Against Workers

After doing some performance testing with thousands of transactions, I've decided that the performance hit isn't eggregious enough to warrant adding in workers.

As such, I've decided to just put a loading state on the Next button when transitioning to the Adjust Transactions step. UX-wise, that should be good enough.

Supported Fields

As much as we can technically have something like a rule that modifies the amount, I don't think we *should*. Like, how often does that actually come up? It just muddies up the options and increases cognitive burden, so we should only support the most common use cases.

That means the only supported fields are description and account (for conditions) and description, account, and type (for actions).

Remember, keep the surface small to start with, to address 80% of the use cases.

How does Ranking Rules actually work?

From Import Rules System, "Brainstorming → Rule Table Item → Rank":

Maybe 'rank' is like a measure of complexity? So that a low 'rank' just indicates a low complexity (i.e. a very general condition) whereas a high 'rank' indicates high complexity (i.e. a very specific condition). In which case, how do we calculate complexity? Number of conditions * number of characters in the condition (e.g. 1 condition with a text input value of 'test' would have a 'rank' of 4, but 2 conditions with text inputs values of 6 and 8 = 2 * (6 + 8) = 28). But then 'ranking' the rules is just putting them in order of complexity.

How to Toggle Rules on/off during Adjust Transactions

If we want to support toggling the rules on/off during Adjust Transactions, how the heck are we going to do that?

The most obvious problem is what do we do after a user has made a modification? Like, if they modify it with the rules off, then re-enable the rules, what's supposed to happen? If they modify with the rules on, but then turn the rules off, what happens?

Well first of all, how do we even represent rules on vs off? The first thing that comes to mind is having two transaction maps in the import state: one for without rules, one for with.

Then... I guess if users make modifications to on or off, it just makes modifications to that particular version? That seems like the simplest solution, and it's not like users will be regularly toggling rules on and off. So... should be fine.

- Although maybe it'd be good so that modifications to fields that aren't affected by rules (e.g. notes, amount, date) are mirrored across both states.
- Or maybe all user modifications should be mirrored across both states, cause obviously whatever the user overrides should take precedence over what a rule changes? i.e. user modifications have a rank higher than all rules?

Yeah, I think we should just mirror user modifications to both states. Keep things simple and treat user modifications as the final 'rank'.

Then the transactions table/list just pulls from one state or the other based on the toggle state. Simple enough.

Import Profiles on Overview Page

I just had the most brilliant idea. Now that we have the Import Overview page, it gives us a logical spot to manage the Import Profiles! No longer do we need to throw them into the Settings page!

Of course, adding Import Profile management is utterly out of scope of this ticket, but it's something good to keep in mind.

Editing Rules during Adjust Transactions

Currently, we've allowed the design to include toggling and adding rules during the Adjust Transactions step, but do we want to support editing the rules as well?

Strictly speaking, it *is* a better UX. But it makes sequencing things more complicated. Like, if we change a rule, doesn't that mean that we need to then re-run all of the rules against all of the transactions, since the ranking could have changed?

And if we *do* have to re-run the rules, do we re-run them against the transactions as they were coming into the step (i.e. the 'original' transactions)? Or do we re-run them against the set of transactions that doesn't have the rules applied (i.e. including user modifications)? I think it *should* be the latter, because (again) we want to preserve user modifications as much as possible.

And taking this logic also means that it applies to adding new rules. That means, whenever a rule is added or changed, we need to use the user-modified no-rules-applied set of transactions. and we need to re-apply all of the rules to all of the transactions.

Yeah, we're *definitely* going to need to parallelize this work using Web Workers.

Handling Conflicting Conditions/Actions

What happens when a single ImportRule has, itself, conflicting conditions? Or even conflicting actions? Like, if a rule has two conditions that target the description, what do we do?

I think the simplest solution is just to prevent a rule from having overlapping conditions/actions. Once a user has selected a given property for a condition/action, it should not be allowed to be used in another condition/action.

Input Validation

Don't forget that we need to have a maximum on the length of the values for the conditions/actions.

I'm thinking the same length limit as for transaction descriptions?

Updating Import Rules

It has come to my attention (as part of implementing the sagas) that we don't actually have an existing pattern for updating an object and its associations at all once (cause we never implemented `update` for Import Profiles).

That is, the problem is, during the commit/effect/rollback phases of a request, how do we update the actions/conditions of a rule when there is the possibility that existing actions/conditions were either modified or deleted, or new actions/conditions were created?

I think we need some sort of diffing algorithm that spits out the following:

- IDs for created
- IDs for updated
- IDs for deleted

That is, it'd take the list of action/condition IDs from the original rule, diff it with the new rule, and create the above sets of IDs.

- New IDs are those that exist in the new rule but not the old rule.
- Updated IDs are those that exist in both rules.
- Deleted IDs are those that exist only in the old rule but not the new rule.

This is what would happen for each phase for each set of IDs.

Commit

- For new IDs, the commit phase should add the new actions/conditions to the store.
- For updated IDs, the commit phase should update the actions/conditions in the store.
- For deleted IDs, the commit phase should remove the actions/conditions from the store.

Effect

Same as Commit, accept make the Feathers calls rather than store calls.

Wait, nope, that doesn't work at all because we don't expose the action/condition services. Does this mean that we *will* have to expose them? Or can we have the rule service handle the diffing? It'd just have to lookup the old rule and apply the same diffing algorithm.

I had considered modifying the `update` method so that it could also do `create` if the ID didn't exist (i.e. `upsert`), but meh.

Rollback

I think this is the first time that we'll need to use the `originalPayload` property of the rollback data. Why? Cause we'll need to compute the diff'ed IDs again. And then operate on IDs from both the old and new rules.

- For new IDs, they need to be removed from the store.
- For updated IDs, they need to be re-updated with the data from the old rule.
- For deleted IDs, they need to be added back to the store.

Form

We're gonna need to use a new feature from `react-hook-form` to implement the ImportRuleForm: `useFieldArray`. This is basically the function to support arrays as fields (go figure).

This code sandbox is a decent example: React Hook Form - useFieldArray - CodeSandbox

Active Import Rules

A thought occurred to me... for the "active" import rules in the Adjust Transaction Step, we give the user the ability to add new rules. But, how do we show these new rules in the list of active rules?

In theory, they should only show up in the list if their active, but the user might get confused if they create a rule that isn't active. So either we need to show active + new rules, or we need to have some sort of other indication that new (but inactive) rules were actually created (but aren't active).

Re: Pagination

I have chosen not to paginate the Import Rules list/table. I feel like users won't be creating *tons* of rules (i.e. hundreds), so we should be fine.

Re: Outline Button vs Link Button for adding actions/conditions

In the `ImportRuleForm`, the original design used Outline Buttons for the "Add Action" and "Add Condition" buttons.

However, I'm starting to think that maybe it'd be better to use Link Buttons with a "+" icon, ala how the onboarding lists for accounts work.

Haven't made up my mind yet, but I'm just noting down that I'm having these thoughts.

'Contains' Conditions

For `contains` conditions (which ultimately come down to an `includes` string call), what kind of string processing should we include?

For example, our transaction description indexing process includes things like lowercasing everything so that the search process is case insensitive. Should `contains` conditions also be case-insensitive?

Should they also go as far as ignoring things like stop words?

I think we *should* make them case insensitive, but I *don't* think we should make them any more complex than that. That's what the regex conditions are for.

Integrating Rules into the Import Process

- Add a new "areRulesActive" state piece to the import process state.
- Add a new "transactionsFromRules" state piece to the import process state.
- Create a cross-slice `transactionsImport` selector that selects the active rules.
- Create the service for applying rules to transactions.
- Modify the `parseCsvIntoTransactions` import saga to apply the rules after marking duplicates.
  - This should create a new set of transactions that goes into the `transactionsFromRules` state, whereas the set of the transactions before applying the rules goes into the regular `transactions` state.
- Add the `ActiveImportRules` to the `AdjustTransactionsStep`.
- Modify the `AdjustTransactionsStep` `connect` to pull the right transactions based on whether the rules are active or not.
  - When rules are on (the default state), pull from `transactionsFromRules`.
  - When rules are off, pull from `transactions`.
  - Should encapsulate this logic into a selector at the slice level.
    - This way, all sub-selectors inherently choose from the right state.
- Modify `updateTransaction` action of the `transactionsImport` slice to update transactions in both the `transactions` and `transactionsFromRules` states.
  - This way, editing a transaction mirrors the update across both states.
- Modify the `cleanTransactionsForSummary` saga to pick either `transactions` or `transactionsFromRules` based on `areRulesActive`.

    - It just works because of modifying `selectAllTransactions`.
        - That was really smart to think of.
- Add an import saga that listens for new rules or updates to rules and re-applies all of the active rules.
    - Since it listens for all rule changes, it should have a gating condition that so that it only runs during the Adjust Transactions step.
    - To re-apply the rules, it should lookup the active rules and then apply them to whatever is in the `transactions` state, to then update the `transactionsFromRules` state.
        - This way, user updates should be preserved.
        - It also has the side effect of allowing users to manually edit transactions and then apply rules to them; whether or not that is *useful* is another matter.
- When adding a rule from the Adjust Transactions step, a warning toast should be issued whenever the rule isn't considered an 'active' rule.
    - This way, users will know that the rule was created but not applied.
    - Don't need a (success) toast when creating a rule that gets applied, since it will just show up in the list and the user will see it.
- Need to change the text input to a select input for actions with the `Account` property.
- Ditto of the above for the `Type` property.
    - I wanted to use a `TransactionTypePicker` for this but realized it'd be a rather poor fit.
        - Mainly because of the color scheme (dark on dark), but also because there just isn't horizontal room for it.
    - So... regular old `SelectInput`.
- Modify the rule table row/list item to pull in account names for the Account properties.
- Implement account/type properties in the rules applier.
    - In order for this to work properly, the `ImportRulesApplier` service will need to take in the account that is being imported to, and the interface will need to change to accept `ImportableTransactions` rather than regular `Transactions`.

    - This is so that we can read the `targetAccount` property from the `ImportableTransaction` to match it against the raw text value of an Account condition.
    - And then so we know which of the debit or credit accounts to rewrite with the action's account ID.
  - Actually, scrap that, our current logic is just "account type !== expense ? put target in credit, else put target in debit".
    - As such, we could just not pass in the account we're importing to, and instead just derive it like so.
    - That does, however, mean that we should encode that logic into an ImportableTransaction model function, so that the logic is shared between the applier and the `TransactionForm`.
  - Actually, scrap *that*.
    - Turns out, we *do* have a function for determining which account is the 'target' account: `ImportableTransaction.determineTargetTransactionSides`.
    - The logic in the `TransactionForm` (which was somehow based on the Expense type) has been updated accordingly.
  - Additionally, it must be noted that a rule with a type and account action *must* apply the type change before the account change, due to the above logic.
    - That kinda means that we need an "action sorting logic" so that actions get sorted before being applied.
- Add validation to ensure account/type actions don't create invalid transactions.
  - Account properties are going to need to be validated, since a user can pick any account for the rule, but the chosen account might not make sense for a transaction.
    - What do we do here? Just ignore the action? I don't want to have to like annotate the transaction row/item with a note saying "well we tried, but your rule sucks".
  - Type properties *also* need to be validated to ensure that the type is valid given the account that it is being imported to.
    - So we *will* need to pass in the import account to the applier.
    - Wait a minute, shouldn't the import account already be attached to the transactions by this point?

        - Yes, yes it should.
        - So we don't need to pass it in separately. Hooray!
- The only problem with doing this level of validation at the applier is that we don't have a meaningful way to surface these errors to the user.
    - So we'd either need to add some sort of 'rule errors' section to the Adjust Transactions step.
    - Or we just ignore the errors.
    - Or we don't validate anything at all until the user tries to move to the Next step in the import process, at which point all transactions will be validated and an error message can be show then.
        - We *should* be doing this regardless however. The current process just assumes that the user can't create an invalid transaction, but now that they definitely can, we should add like an error message above the Next button or something.
    - Or do like we do with bulk editing and just wipe out the import account from the transaction if we create an invalid transaction.
        - Basically, just make sure the final transaction is valid and force the user to fix things themselves manually.
        - Just don't get put into a spot where a transaction is invalid.
            - Might be more annoying for users ("why wasn't my rule applied??"), but easier for us.
    - Or, if any action would create an invalid transaction, then we don't apply that rule *at all* and then retroactively mark it as a rule that isn't 'active'.
        - With this approach, we eliminate some of the confusion for a user around "well, it says the rule is active, but it seems to have only applied parts of the rule; why didn't it apply the rest of the rule??"
        - However, this approach also causes the feedback loop for the user to be diminished since inactive rules can't be brought up and edited easily in the Adjust Transactions step; the user would have to go back to the Import Overview to fix them.
        - Plus, they would likely wonder *why* a rule wasn't active to begin with.
            - They'd be like "well, the conditions were met, so why isn't the rule active?"

        - So... we'd probably need some sort of separate "would be active but aren't" rules section, or even just a separate list of all the inactive rules.
            - If we went with "would be active but aren't", we'd probably have to specify *why* the rule isn't active. Which would be the better UX.
            - But the easier solution development wise is just tacking on the list of inactive rules and letting the user figure it out.
                - But I can definitely see this being a frustrating experience for the user.
        - And what about weird cases like a rule that successfully applied to one transaction but not another. How on earth would we handle that??
- I have decided on the following logic:
    - If a type action can be applied given the transaction's import account, do it.
    - If it can't, then apply the type anyways, but wipe out the import account.
        - This is to mimic how the bulk update system works, where changing types to something the import account doesn't support just wipes out the import account, forcing the user to manually apply it.
    - If an account can be applied given the transaction's target account, do it.
    - If it can't, then ignore the account action.
        - This is done because we can't exactly force account actions to change the type to accomodate the account when we've already decided that type actions have precedence over acount actions.
- As a result of the above logic, we will not be showing any further error messages or anything to the user.
    - Rules will just be applied on a 'best effort' basis, so partially applied rules are definitely a possibility.
    - As long as the output of the rules applier are *valid* rules (within the context of the Adjust Transactions step), then we should be fine (at least, for a first pass at implementing a rules system).

- When a rule that has an action on an account has its associated account removed, fix the rule somehow.
    - I guess the most logical solution would just be to remove the action from the rule?
        - This makes more sense than just... keeping the action but nulling the value.
    - Somehow, I think this is going to need its own offline request slice... since it's the type of request that would require rollback if the original required rollback.

Extra Validation

We need extra validation more validation during the Adjust Transactions step to ensure that a transaction doesn't have an invalid mix of accounts set.

I thought this was only theoretically possible yet it only took me like 30 seconds to come up with a case where it can happen:

i. Say we're importing to an asset account called "Cash".

ii. Say we have a transaction that is initially of Income type.

iii. Say we have a rule that changes its type to Expense.

iv. Say we then assign the transaction a target (expense) account of "Food" using the bulk update buttons.

v. We now have a valid Expense transaction with valid Cash and Food accounts.

vi. However, say we now *disabled* the rules, such that we're now looking at the transaction without the rule change (i.e. Expense type change) applied, but still with all of the user-specified changes applied.

vii. We now have an Income transaction with the Food expense account still applied, but missing the Cash account.

viii. We can now assign the transaction a target (income) account using the bulk update buttons.

ix. BAM, a super totally invalid transaction that has an Income and Expense account on it.

Our current validation would not prevent this transaction from being passed on to the final step.

As such, this transaction should be marked as 'invalid', likely using the same `whyTransactionsArentValid` and `transactionsAreValid` methods that are already present. We'll have to pass in the `accountsById` to the validation functions so that we can check the types of the accounts.

We should also improve *how* we display these validation messages to the user. Although the Next button is disabled when there is an invalid transaction, we should add a click handler that shows the validation message above the buttons when the user clicks on the button *while it is disabled*. That way, when the user gets frustrated and tries to mash the button, they'll see the error message.

Update the Backup Format

- Increment the file version.
- Update the changelog in `BackupService`.
- Add the `importRules`, `importRuleActions`, and `importRuleConditions` to the format.
  - Update `parseBackupFile`.
  - Update `validateData`.
  - Update `_validateFile`.
- Add tests for the new objects.
- Add tests to ensure the old formats still work.
- Update the `selectDataForBackup` selector.
- Update the user sagas.
  - `restoreBackupCommit`
  - `restoreBackupEffect`
  - `encryptOrDecryptData`
- Update the file version in the e2e tests.
- Update the backup files used for the e2e tests.

ImportRule Model

We need three new data models: ImportRule, ImportRuleCondition, and ImportRuleAction.

- ImportRule has-many ImportRuleCondition.
- ImportRule has-many ImportRuleAction.

ImportRule fields

- id
- userId

ImportRuleCondition fields

- id
- importRuleId
- condition
- property
- value

ImportRuleAction fields

- id
- importRuleId
- property
- value

What we Need

- Import Overview page
  - Import Options
  - Import Rules List
- Import Rule Form (for creation/editing)
- A modification to the Adjust Transactions page to include the active rules, a way to toggle rules, and a way to add rules
- A modification to the Adjust Transactions step logic to apply the rules
- A modification to the import process so that we can toggle the rules on/off
- An ImportRule model
  - ImportRuleCondition model
  - ImportRuleAction model
- Store slices
- Add the models to the encryption middleware.
- ImportRule sagas (CRUD)
- A modification to app boot to pull all of the above models.
- A service that takes in a set of ImportRules, a set of Transactions, and applies them

    - Although, this 'service' could just be a/some functions on the ImportRule model, ala how we do it with the ImportProfile.
    - Although, the fact that we kinda need to decouple it (for API client purposes), not to mention the need for Web Workers, kinda makes it seem like it'd be better off as a standalone service.
- A modification to the Backup schema to include all these new models.
- Backend:
    - ImportRule schema, table, and service
    - ImportRuleCondition schema, table, and (private) service
    - ImportRuleAction schema, table, and (private) service
- Marketing:
    - Update changelog.

Component Breakdown

Atoms

- FormCardContainer
- LargeButton

Molecules

- ImportOption (LargeButton)
- ImportRuleCondition (FormCardContainer)
- ImportRuleAction (FormCardContainer)
- ImportRuleListItem
- ImportRuleTableRow

Organisms

- ImportOptions (ImportOption)
- ImportRuleForm (ImportRuleCondition + ImportRuleAction)
- ImportRulesList (ImportRuleListItem)
- ImportRulesTable (ImportRuleTableRow)
- CombinedImportRulesView (ImportRulesList + ImportRulesTable)

- ActiveImportRules (CollapsibleSection + CombinedImportRulesView + ShadowButton)

Scenes

- ImportOverview (ImportOptions + CombinedImportRulesView)
- SidebarImportRuleForm (ImportRuleCondition + ImportRuleAction)
- [modification] AdjustTransactionsStep (add ActiveImportRules)

Tasks

- Build the Backend.
- Build the Frontend.
  - Update Marketing changelog.