Story UFC-207: Dashboard Account Summaries

The following document will go over the design of the dashboard account summaries, as outlined in Story ticket UFC-207.

Acceptance Criteria

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

- On my dashboard, I want to see the current totals for each of the four account types.
- I want to be able to control which date range is being viewed for the account summaries.
- For Assets and Liabilities, I want the 'From' amount to be calculated as everything from the beginning of time till the start of the current date range.
- For Income and Expenses, I want the 'From' amount to be calculated as everything in the interval previous to the current date range.
- I also want to see the percentage change (and difference) between the current and from amounts.

Design Brainstorming

Account Balance Date Index

OK, so we were originally going to implement this new index for indexing account balances by date, so that we could speed up the calculations for the Asset and Liability summaries (since they need to be calculated from the beginning of time).

However, I've now decided that this is overkill (at least, for now). Instead. since we introduced using `re-reselect` in UFC-195 (transaction type filtering), I think we can just leverage it again to cache the account balances.

So really, that means we just need to focus on creating new selectors for calculating the account balances in date ranges:

- Asset and Liabilities: beginning of time till start of date range + current date range
- Income and Expenses: previous date range + current date range

This should be pretty simple considering we already have a selector for calculating the balances by type (that was used for the old account summaries), so there shouldn't be

much logic work required for this story (I think taking the simple route here is what's called 'making the smart startup decision).

I think we'll need 2 selectors:

i. Calculate all balances by type between two dates.
   - `selectBalancesByTypeBetweenDates`

ii. Calculate asset and liability balances from beginning of time till a date.
   - `selectAssetLiabilityBalancesToDate`

Selector Implementations

OK, so I've realized that *maybe* it's not completely trivial to implement these selectors...

So the current way we go about calculating the (old) account summaries is to look up all the accounts, separate them by type, then sum each account's transactions to generate the balance, then sum each type's accounts' balances to get the balance per type.

But... this leverages the fact that the accounts get populated with the full transaction objects as part of one of the higher up selectors.

So then how do we go about injecting the transactions date index into this to 1. speed up querying and 2. select the right range of transactions (note that we old logic actually just did a linear scan over *each* account's transactions to filter them by the current month to generate the account summaries... bleh).

First I thought "well shit, maybe we *will* need the account balances date index". But then I thought "no, there *must* be a simpler way".

One of the ideas I considered was some sort of selector that queried an individual account's balance between two dates, but then I couldn't figure out how to then leverage this in a super selector to select *every* account's balance between two dates (cause that's not really how selector composition works).

But then I had the idea of, what if we completely change how we do balance calculations? Instead of relying on the populated transactions in the account object, what if we queried all the transactions between two dates and then used those to compile the balances.

That is, we do something like this:

i. Get all transactions between two dates, and get an accounts by ID map.

ii. Loop over the transactions. For each transaction:
   1. Take the IDs of the credit/debit accounts from the transaction.
   2. Take the amount of the transaction.

3. In the accounts by ID map, using the credit/debit account IDs, lookup the accounts and add the amount to the 'balance' of the account.

iii. Add the 'opening balance' of each account to its current 'balance' (assuming the 'balance' property starts at 0).

This will get us a map of all accounts with correct balances for a given date range.

Then, we can collapse this into a map of types fairly easily (create an empty map of types, loop over each account, add each account's balance to its corresponding type).

So this solves the problem (non-optimally) of getting balances by type between two dates.

This algorithm can then be used for generating the balances by type for assets and liabilities from the beginning of time till the start of the date range (we just modify which dates we use when querying for the transactions) as well as the balances by type for income/expenses in the previous date range (again, just need to modify the dates).

There is one extra note here: should we hardcode 'the beginning of time' as an actual date (e.g. 1970) or should we have a separate date index selector that accepts just one date and does the 'beginning of time' lookup itself? Or, should we just modify the existing 'between dates' selector to accommodate the special cases of one of the dates being an empty string:

i. If the start date is an empty string, then select everything from the start of the index will the end date.

ii. If the end date is an empty string, then select everything from the start date till the end of the index.

I kinda like that modification option, since it reduces our API surface area.

But there's still something else of note: currently, the 'between dates' selector includes both the start and end dates when looking up transactions.

However, when doing the beginning of time till start date lookup, we *don't* want to include the start date. So should we have an options argument for making the selector include or not include the start/end dates, or should we have the logic moved up to the caller of the selector to manually subtract a date before selecting?

I think we should go the options route, just in case we need in the future.

- **Update**: I went the lazy route and just did the subtraction in the hook. If we need to do this again, then I'll reconsider moving the logic into the selector.

OK, so I think that solves our problem. We should now be able to perform a "between dates" selection for "balances by type". We just have to do it three times:

i. Get all balances by type between the given date range.

ii. Get asset/liability balances before the date range.

iii. Get income/expense balances in the interval before the date range.

So ultimately, I think there'll only be one selector: `selectBalancesByTypeBetweenDates`.

It's in the hook that we perform the three above selections, before combining them together to get the final 'account summaries' view.

Again, this is all very *non-optimal*, but until we start hitting performance problems, I think it's fine to forego the account balances date index for now.

Account Summaries Organism

Honestly, there shouldn't be too much to do for this component. We've already built all of its building blocks, so now it's just a matter of slapping them together.

The most notable thing is that the component will directly leverage whatever hook we create to combine all of the selectors created above (I'm thinking we call it `useDateRangeAccountSummaries`).

Dashboard Scene

Wow, we *finally* get to put something here!

We've touched it so little that we haven't even put the OverlineHeading in it yet!

In terms of other things, there's not much to it. We'll need to add in the `DateRangeProvider`, then add in a DateRangePicker, then finally add in the new AccountSummaries.

Since we won't be passing anything down anywhere from the Dashboard scene, we won't need a crazy wrapper setup like the Transactions scene; in fact, I think we'll only need the one component, with the DateRangeProvider in it, since we won't need to access the date range stuff in the Dashboard scene itself.

Overall Design

Overall, a fairly straight-forward story. I think I find it to be straight forward because we don't have any new *interactive* elements; it's really just combining a bunch of existing components and some new *display* only components Hopefully this kind of thing becomes more common as time goes on, as these are nice tickets to work on.

Component Breakdown

Atoms

- N/A

Molecules

- N/A

Organisms

- AccountSummaries (AmountChange. CurrentAmount, FromAmount, Divider)

Scenes

- [modification] Dashboard (AccountSummaries, OverlineHeading, DateRangePicker)

Tasks

- Build the new date range account balance selectors.
- Build the hook for getting the account summary data (`useDateRangeAccountSummaries`).
- Build the `AccountSummaries` organism.
- Modify the `Dashboard` scene to include the `AccountSummaries`.