Story UFC-184: Accounts List

The following document will go over the design of the accounts scene, as outlined in Story ticket UFC-184.

Acceptance Criteria

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

- I can click on 'Accounts' in the header to go see an overview of all my accounts.
- On the 'Accounts' page, I can see the names and balances of all my accounts.

Design Brainstorming

I don't think there's really much to be discussed in terms of what this new version of the accounts list will do vs the old one. We pretty much use the same data structures (accounts grouped by type), so much of the redux store logic can be reused.

Obviously the interface is completely different, being much more of a list than a table. But overall, it should be fairly simple to implement.

Just to note scope for this story, as far as the Accounts page is concerned, we're *only* dealing with the list of accounts; not the type picker, not the search bar, not the add button, not the date range picker, and certainly not the rest of the Account Details view.

However, what *does* concern discussion is how we're gonna deal with the fact that the Accounts scene is two different layouts based on screen size: there's the mobile accounts list where only the list is shown, and there's the desktop accounts view where a (differently styled) list is shown along with the account details.

I feel that the we'll have to make the mobile list a completely separate component from the desktop one, because they're just styled so differently. Additionally, because the mobile accounts list has much of the same stylings as the mobile transactions list, we'll have to decompose it accordingly. I'm thinking the base list item (just the container with the overflow actions) can be an atom, whereas the mobile accounts list can be a molecule, with the desktop accounts list another molecule, and the final accounts list that combines them together can an organism.

Tab Bar

This component is.. interesting. Because of the way the sliding underline is implemented (a separate element whose position and size are transformed using `translateX` and `scaleX`), we either have to set all tabs to be the same width, or to see all tabs with an explicit height before hand so that we can calculate the correct amount of translation and

scaling. Plus there's the whole problem of things shifting around when text becomes bold, so then we have to account for that in the width calculations or fake the bold with `text-shadow`.

Obviously, neither of these is optimal, i.e. automated.

But what if we could automate it? We'd certainly need some JS.

I'm thinking an automated `TabBar` component could do the following:

- Take in an array of tabs. This can have either string labels or custom components for rendering, doesn't matter.
- Attach refs to each tab component.
- Attach a ref to the tab bar container.
- Use the refs to get the rendered width from `offsetWidth`.
- To set the underline position and size, get the width of the active tab and the width of the container and calculate the translation and scaling amount using the formulas we already have. Then attach inline styles to the underline component.
    - We'll have to remember that we'll need to prefix the `transform` properties ourselves.

It might just work.

List Item

I've kept waffling back and forth over what the ideal design is for these stupid things.

Initially I wanted to have separate components for the Small and Large versions (i.e. mobile and desktop), but then I realized that, like the Logo, I could just have the two versions under one component with a flag.

But then I really didn't like the small/large differentiation between them, because it seemed kind of arbitrary. Yes, *currently* one design is used on mobile and one is used on desktop, but that might not always be the case. Additionally, we've already established with things like the ShadowButton and OutlineButton that *semantic* variations are OK.

Therefore, I've decided to go with the semantic variations of 'double layer' and 'single layer' list items. So named because the mobile versions have two layers (the top with the content and the bottom with the actions), whereas the desktop versions have just the one layer with the content *and* actions exposed. I think keeping these as one component with a prop to differentiate should be fine for now (if somewhat inconsistent with the examples I just gave above).

Additionally, I just want to note that it'd probably be a good idea to build out a specific `AccountsListItem` molecule that binds the content layout but also exposes a *connected*

component that only takes in an ID. This is to go along with the learnings we had with the Transactions table in the old design, whereby it was *so much* more efficient (performance wise) to just pass IDs and connect the rows/items. Then the `AccountsList` can just map over a list of IDs. Easy peasy.

Component Breakdown

Atoms

- IconButton
- ListSectionHeader (needs to be responsive for small + large)
- LinkButton

Molecules

- ListItem (with single/double layer variantions)
- AccountsListItem (a ListItem bound with the content of an account and connected)
- NavItem (the icon + TextField needed for both the small and large nav bars)
- TabBar

Organisms

- AccountsList (SmallAccountsList + LargeAccountsList)
  - I decided that the small/large versions don't really need to be separate molecules, since they aren't going to be used anywhere else. So we'll just keep them together under this one organism (just like what we're doing with AppNavigation).
- [modification] rename AppHeader to AppNavigation (then add in the NavItems)

Scenes

- Accounts (OverlineHeading + AccountsList)

Tasks

- Build the navigation for the AppHeader (including the mobile view). **UFC-216**
- Build the basic Accounts scene. **UFC-215**