Freighter Tiered Performance Improvements

OVERVIEW

Freighter's UI still feels a bit sluggish, especially when navigating back and forth between the Account view (main view) and other views like History, Send, etc. This is a persistent issue and must be addressed before we continue to add new features as we are turning users away with poor UX before we're able to show them what Freighter has to offer. As with any application, there are optimizations we can make both in the UI layer as well as in the backend.

Upon looking more closely into these issues, I discovered there are some obvious "quick win" improvements we can make as well as some issues into which we must dig further. There are some UI patterns with clear issues, but there are also some unsolved mysteries around slow load times for a subset of users. We will create a plan to address the issues with clear answers right away while simultaneously gathering more context about the unknown issues so we can quickly follow up and solve those, as well.

First, we'll dive into Freighter view by view. For each view, we'll dive into small, medium, and large t-shirt size improvements we can make to each. We'll discuss each improvement in relation to the baseline loading time I'm seeing for my test account (GBTYAFHGNZSTE4VBWZYAGB3SRGJEPTI5I4Y22KZ4JTVAN56LESB6JZOF) on Mainnet on my M4 laptop on my home wifi network in NYC. This account has 15 trustlines and 2 SEP-41 tokens. Although this load time may not be typical, it will at least provide us a baseline from which to extrapolate.

A metric we will look at frequently is Largest Contentful Paint (LCP). Generally, this means the amount of time it takes to go from page load, past the loading spinner, to a view that shows the user actual data with which they can interact. Our goal is to get that number below 2 seconds in all cases.

Second, we'll discuss observations about our backend response times. To dig deeper into slow response times, I'd like to do an additional design spike working alongside a backend engineer. As a starting point, I'll lay out some trends I've observed and some theoretical solutions to explore. In addition, I've listed some actual concrete optimizations that we can start to think about implementing.

BACKGROUND

This document presumes that you have read through this previous doc related to this topic. This doc does a great job defining our pain points and our overall approach to loading pieces of UI blocking data like balances. Below, I'll start to drill deeper into some of the topics broached in the last doc.

For additional context, we'll need to look deeper at how we load asset icons in Freighter, as this can also be a very costly request. The first time a user ever loads a trustline in Freighter, we attempt to load and then cache the asset icon in the extension. Because Stellar didn't have a standardized simple method for web UI's to retrieve asset icons, we implemented asset icon fetching by looking up the asset issuer's home domain. Then, hitting the home domain to retrieve the asset's toml file before finally fetching (and caching) the icon from said toml file. On first load, this leads to an astonishing 3 XHR's to retrieve an icon. If this fails, we look to the user's asset lists for icons. If we still aren't able to find the icon, we retry this method on every page load in hopes that we will finally find an icon. In the optimistic path, we do this painful process once and then cache the icon, never having to make these lookups again. Otherwise, for every asset missing an icon, we do this whole lookup on every single load.

GOALS

1. Identify changes that will improve both perceived and actual loading times
    a. Actual loading time improvements can be defined as optimizing code to actually run faster
    b. Perceived loading time improvements can be defined as changes to the UX that make the user believe all the data has loaded while processes are still running in the background
2. Categorize possible improvements by the size of the job and its relative impact
3. Be able to assign rough estimates to loading time savings

Sizing and prioritization:

For each task, I assigned a rough estimate of the amount of work (t-shirt size) and a priority loosely based on the amount of time needed to complete relative to the improvement we would see.

T-shirt sizing Key:

X-SMALL = 1 day

SMALL = 1-2 days

MEDIUM = 2-3 days

LARGE = 3-5 days

X-LARGE = 5+ days (likely too ill defined to size properly)

Priority Key:

HIGH = small/moderate with high savings

MEDIUM = moderate work with low savings

LOW = high amount of work with low savings

UI OPTIMIZATIONS

Account

This is the most high priority view to improve for multiple reasons. First, this is the main view a user lands on when opening Freighter and also the default screen a user gets kicked back to after completing a flow (like Send, for ex). Because of that, slow performance here can immediately sour a user on their overall experience of the app. Second, this is one of our slowest loading screens because of how we fetch balances AND how we fetch icons. If we can find improvements here, we can extrapolate this to other views that need to load the user balances.

One of the main bottlenecks here is the amount of XHR requests we make before LCP. Currently, not only do we load the `/account-balances` endpoint from Freighter BE, we also load `/history`, token lists, and various G accounts from Horizon (because of the aforementioned missing asset icon lookups) before we show the user anything other than the loader (as seen in the example video).

Current baseline: I'm observing the largest call being the `/account-balances` endpoint that takes about 800ms. Next up, each around 300ms each, are `/history`, and perhaps surprisingly, LOBSTR's asset list.

Generally, for my test account, this view takes 3-4 seconds for LCP

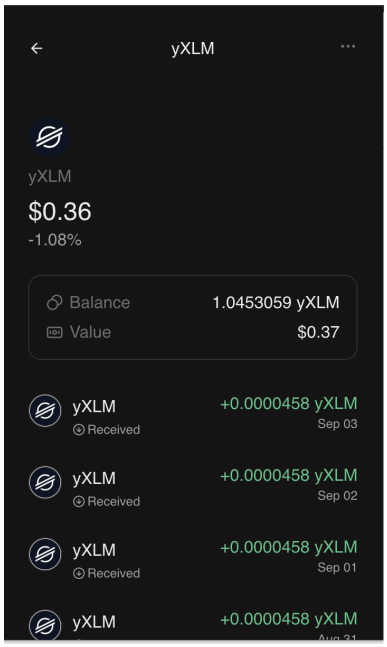
In the below video, I have throttled the network to a slow 4G speed so we can illustrate all of the XHR's we are waiting to resolve before loading any information for the user:

account-loading.mov

Load history in the background

T-shirt size: SMALL:

One of our biggest requests is for `/history`, the response of which actually is not seen on the Account view. We need history when we click onto an individual asset, which takes the user to the Asset Detail subview. This subview is where we actually see the history response as this view shows history items related to that specific asset.



Because we don't actually need this data right away, instead of blocking the view until history is loaded, we can simply load this history in the background.

This reduces my LCP by about .5s, the amount of time needed to load my account's history

UX IMPACT: Hopefully, by the time a user clicks on an asset row, this data has finished loading. But in the worst case, a user may have to wait an additional number of milliseconds for their Asset Detail history rows.

Priority: HIGH

Optimize Webpack bundling using tree shaking and caching

T-shirt size: MEDIUM:

Webpack, our current JS bundler, does offer a significant amount of customability when it comes to bundling your code. By removing dead code and slimming our JS bundles, we can conceivably improve load times. In my test environment, I'm already seeing quite fast load times for JS files as the files are being loaded from the user's machine and no XHR roundtrip is needed. In my opinion, this is a complex task that may only save us small amounts of time.

This would conceivably only reduce my LCP by about 100ms

UX IMPACT:

None

Priority: LOW

Poll and Cache account balances

T-shirt size: MEDIUM:

One of the compounding factors for slow load times is that we make a fresh request to the `/account-balances` on every load of the Account view. So, if a user opens Freighter and lands on Account, they have to wait 2.5-3 seconds before they see anything. Then, if they go to Settings and then go back to Account, they have to wait an additional 2.5-3 seconds while we make another request to the account balances. What can be especially frustrating about this is that a user's balance may not have changed, yet we make another request. We have historically opted to always showing the most up-to-date possible information. However, we could possibly take some shortcuts here by strategically showing a cached balance.

One possible solution: In the event that the user has already loaded and cached their balance, upon navigating back to Account, we could show the user their cached balance and in the background, make a fresh request for the updated balance. In the event of a delta, after a brief delay while the fresh request is completed, we could then update the values in the UI.

Another possible solution: In Freighter Mobile, we have a drag to refresh balances feature and there is not an expectation that balances are always up-to-the-minute. We could implement something similar so users have some control over retrieving their own latest information.

In either of the above solutions, we would still opportunistically force an account-balance refresh when the user takes an action that would impact their balance. For example, any time a user completes a send/swap or adds a trustline, we would proactively fetch the latest balance so when they go back to the Account screen, they will have the latest balance without the need for an update.

This reduces my LCP by about 1s, the amount of time needed to load balances, on subsequent page views

UX IMPACT:

This does break a user's expectation slightly as up until now, a user always had their latest balance. With this type of change, a user may face a delay before receiving the latest update. The user also does not get to enjoy these savings until after they've loaded the balance once

Priority: HIGH

Load missing icons in background

T-shirt size: LARGE:

As described above, we hold up the loading process to try to fetch all asset icon URL's before Account load. The icon URL fetching process *can* be very convoluted. If it's an asset like USDC with a defined icon on their configured home domain, it's likely that Freighter cached the asset url and it's quite fast to load the icon.

If it's an asset like BENJI that previously had no icon set, every time we load balances, we see BENJI has no icon and we need to look up the issuer and then look up the home domain to see if they have added it. Because it's possible that BENJI at some point will add this icon, we want to keep checking. However, we don't need to hold up loading balances while we make this check.

If we still can't find BENJI's asset icon, we check the user's asset lists for an icon. This can also be costly as we need to load the user's asset lists and then check them. In my test account with only the 3 default asset lists, this took about 800ms total. Similar to the home domain look up, we do not need to block loading for this. We can do this in the background.

Furthermore, we can re-order our priorities here: first checking the cache, then the user's asset lists, and then finally the costly Horizon lookup. Caching the user's asset lists will also prevent having to do a fresh lookup on subsequent Account view loads. If an icon cannot be found on this initial lookup, we can simply stop checking on subsequent loads.

Moving home domain lookup for each asset to the background reduces my LCP by about 1.25s

Moving asset list lookup to the background reduces my LCP by about .75s

UX IMPACT:

This experience will greatly vary by user, but one first load, a user may see their balance appear while icons show a loading state as they are initially fetched and then cached. On subsequent loads, their icons should appear very quickly.

Priority: HIGH

Manage Assets

Similar to the Account view, this view suffers from the complex account balance and icon fetching above. Implementing some or all of the changes above will carry over to this view, as well. One additional feature slightly slowing down performance here is that we need to fetch home domains for each asset. Though we do cache these results, on initial load, we are making a call to each asset's issuer on Horizon at least once.

Generally, for my test account, this view takes 3-4 seconds for LCP

Load home domains in the background

T-shirt size: SMALL:

Similar to loading account history in the background, we could proactively load an account's home domains in the background in the Account View. If we did this, a user's home domain cache would likely already be populated once they clicked the Manage Assets button.

The reduction in LCP is nominal for my test account, but could be significant if an account has many assets. For my test account, I observed if each home domain lookup took about 40ms, so savings for my account would be about 40*n ms.

UX IMPACT: Hopefully, by the time a user clicks on manage assets, this data has finished loading. But in the worst case, a user may have to wait an additional number of milliseconds for the asset home domains. Another caveat here is that if a user had Freighter in fullscreen mode and refreshed their browser on the /manage-assets view, they would need to wait for the cache to be populated as that would not have been done already by a previous screen.

Priority: MEDIUM

Send/Swap

Similar to Manage Assets, this view suffers mostly from an unoptimized account balance and icon fetch. The slowest loading view in my observation was loading the user's token list. Making the above optimizations under Account View will improve this experience along the same magnitude of the Account View's improvement.

Generally, for my test account, this view takes 2-3 seconds for LCP

Only make one call to /account-balances on Send/Swap

T-shirt size: X- SMALL:

An additional note here is that I'm observing we're making 2 blocking calls to /account-balances when loading a user's token balance. We should remove this extra call.

The reduction in LCP would be about the length of the /account-balances call, so for this test account, it was about .5-1s

UX IMPACT: None

Priority: HIGH

Sign Transaction

Another view that suffers from blocking calls to /account-balances and icons. Balance is only used to show a user a warning if their wallet doesn't have enough XLM to cover the fee associated with the transaction. The above optimization to load icons in the background would bleed over to this screen.

Generally, for my test account, this view takes 2 seconds for LCP

Load /account-balances in the background on Sign Transaction

T-shirt size: SMALL:

We only use account-balances to determine if a user has enough XLM to cover the fee. This is probably a pretty rare occurrence. Because of that, we may want to opt for loading balances in

the background and showing the user the rest of the tx details in the meantime. We cannot rely on a cache of any kind for this view as Freighter automatically opens a new window to sign a transaction, so we must grab all user info fresh from the BE.

The reduction in LCP would be about the length of the /account-balances call, so for this test account, it was about .5-1s

UX IMPACT: If a user does not have enough XLM to cover the fee, they would get a jarring experience where they suddenly get a warning that takes over the screen after a second or two. We could perhaps mitigate this by animating in this warning rather than abruptly taking over the screen

Priority: HIGH

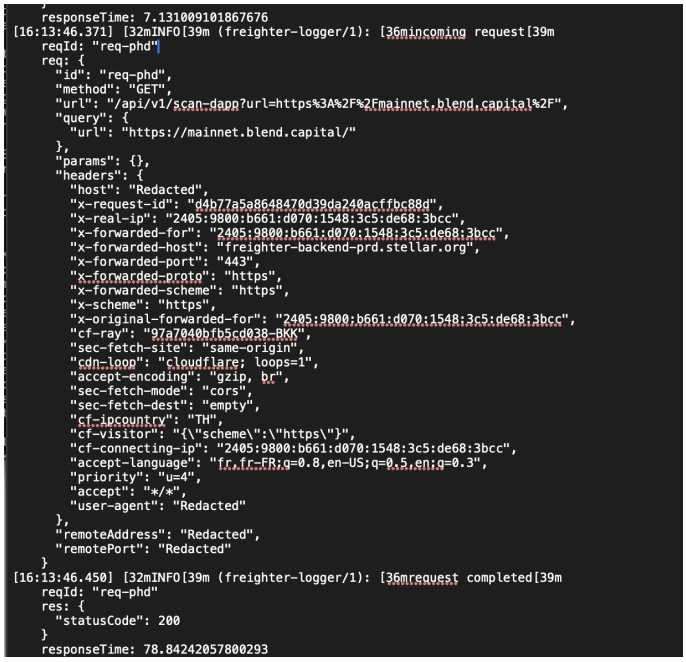
BACKEND RESPONSE TIME OPTIMIZATION

All of our solutions thus far have been based on the assumption that an account with more trustlines and more custom tokens _should_ be our slowest loading wallets. Furthermore, even with an account with numerous trustlines and custom tokens, I have never approached the 30 second wait times some users have reported. We've had some users with only one trustline report these very slow loading times. Although there are some optimizations to be made to the UI, there may be more work to do on the backend.

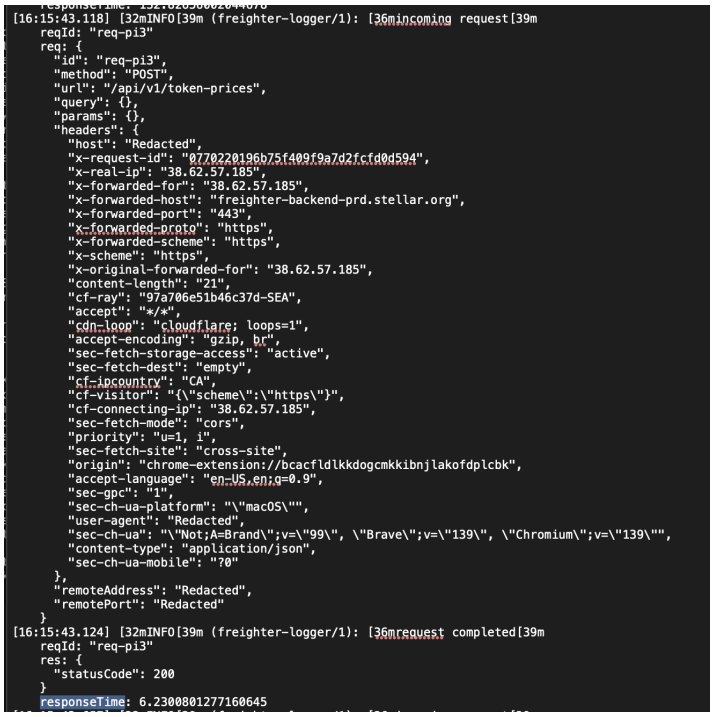
Implement synthetic monitoring to test API response times from various locales at various times of day

T-shirt size: SMALL:

Looking through Argo, we start to see some interesting trends. We see that some very long response times are being reported by users in Brazil, Thailand, and Turkey. This is a small sample size, but can we infer that the user's locale in relation to our data centers is causing some of the user's long load times? Here we see a request from Thailand



We also see some slower than expected load times for a user in Canada, though not on the scale of the user from Thailand:



After talking with the BE engineers, it sounds like we have data centers in US East/West. Perhaps we need to increase our data centers and spread them more globally?

Also, we currently have 2 pods for Freighter BE. Is that sufficient for the number of requests we're receiving? Are there times of day when possibly one pod is getting too many requests, leading to slow response times?

Synthetic monitoring may provide us a more clear picture of where these long response times are coming from. Perhaps this data is already available to us in Grafana?

Priority: HIGH

Implement web vitals monitoring from Sentry

T-shirt size: SMALL:

While we are increasing monitoring for our BE systems, we should also look at increasing monitoring for our UI. After all, we care about the UI loading quickly more than anything else, regardless of API speed.. Sentry provides a web-vitals monitoring tool that we should be able to use to detect if, even after all of our improvements to the UI and BE, we are still experiencing UI slowdowns. Additional context

Priority: HIGH

Use getLedgerEntries to fetch XLM balances and home domains

T-shirt size: SMALL:

Up above, we described fetching home domains (for icons and for the Manage Assets view) and for grabbing XLM balance (for determining if a user has enough XLM to cover a tx). Currently, we do this by fetching an account from Horizon. This can possibly be sped up by using RPC's getLedgerEntries method.

Priority: HIGH

Parallelizing fetching token details in Freighter BE

T-shirt size: SMALL:

When we fetch details about a SEP-41 custom token, we need to get multiple pieces of information: balance, decimals, name, and symbol. Currently, we make these requests sequentially. If we were to make these calls in parallel, rather than waiting for 4 separate XHR's to

complete, we would just be left waiting for the longest of the 4 XHR's to complete. For additional context, we've heard that not many users have custom tokens, so this may be an optimization that only meaningfully helps a small number of users.

For our test account, parallelizing these calls for our 2 custom tokens saved about 250ms in LCP

Priority: MEDIUM

Wallet BE Returns asset icons, home domains, history rows along with balances

T-shirt size: X-LARGE:

We've been talking a lot about fetching/caching icons and home domains in the background to prevent blocking loading pages. This can be fully solved if the `/account-balances` endpoint returns all of the data needed in one payload. This is likely the most complete solution to slow loading times for `/account-balances`.. While this will require a more complete design discussion, essentially Wallet Backend would do the lookup for asset icons and home domains so the UI does not have to worry about fetching the information itself. A possible implementation is for the BE to cache the icon and home domain for every asset:issuer it ever fetches, making the first call for (for ex) USDC a bit expensive, but making every subsequent fetch much faster. The icon and home domains in cache could be periodically checked and updated by a cron job to ensure we're not caching stale information.

Also, we could discuss returning each asset's history rows along with the balances, which would prevent us from having to make a separate call to history and parsing the result for pertinent history rows for each asset.

LCP savings would need a bit more research. This would simplify the Account View as it would just be making one call to /account-balances. This should theoretically be much faster, but unclear if adding these extra tasks to the BE would make the /account-balances endpoint slower

Priority: HIGH

Conclusion

We've looked at a lot of possible optimizations to both the UI and the BE layer. We still have a bit of work to do to pinpoint exactly why some users are facing load times.

In the near-term, we could tackle the high impact changes to the Account View that will immediately make Freighter feel snappier for all users. Meanwhile, we'll start to investigate the BE infrastructure needs and investigate why some users are hitting very long response times. Even if we can't reduce the number of BE calls or their response time right away, we can hide some of those loading times with our UI optimizations. ***We could conceivably move from about 3-4 second loading times to 1-2 second loading times using just our UI optimizations for users getting typical response times from the BE.***

Hopefully after implementing a synthetic monitoring system, we should be able to reduce the amount of very long response times we're seeing from the BE in the near-term. This should bring all users into the ***1-2 second loading time timeframe from above.***

While that work happens, in parallel, the Wallet BE work to return asset icons, home domains in the account-balances can begin. When that work completes, we should have a single endpoint that provides all of our needed information quickly and we can implement that in the UI. ***Optimistically, this should provide the best UX: all information (balances, icons, etc) being available immediately within 1-2 seconds***

Below is an example of how we might tackle the work outlined above.

Beginning Q4 ——
End of Q4

Immediate-Term	Near-Term	Medium-term
1. **Poll and Cache account balances** 2. **Load history in the background** 3. **Load missing icons in background**	1. **Implement any infrastructure changes needed to address findings from monitoring** 2. **Continue monitoring Freighter loading times**	1. **In UI, Implement new Wallet BE routes that return icons, home domains, and balances in one payload**

4. Implement synthetic monitoring of the BE		
5. Wallet BE Returns asset icons and home domains along with balances		

UPDATE 11/17: Freighter Release 5.36.0 - State of the Union