**Frontend:**

Testing the frontend, Lighthouse (Chromium) and other developer tools to determine the metrics below were used. Each test scenario was performed 5 times+ to give an average score.

For testing purposes, a production build of the UI was used, and payments were simulated using mock data. Each payment contained 3 service lines with no adjustments, labels or payors.

While there is no consensus on a limit to DOM nodes, even 10 payments reported excessive DOM size according to Lighthouse. Using developer tools to simulate a weaker CPU 50 payments would slow the page to the point that scrolling fast would outrun the ability for the browser to repaint rows. At 100 payments even without throttling my CPU the page became unresponsive at times and had a noticeably long load time.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Count** | **Size** | **DOM Nodes** | **Time/Speed** | |
| **1gig** | **3G** |
| 1 payment | 4.5 kb | 2,157 |  |  |
| 10 payments | 31 kb | 13,719 | 37 ms | 957 ms |
| 50 payments | 151 kb | 34,366 | 48 ms | 1.67 s |
| 100 payments | 300 kb | 69,908 | 82 ms | 2.51 s |

If we are to introduce a higher payment count the payments grid will need to be modified to use virtual scrolling to maintain reasonable performance. Using react-window shouldn’t be difficult but will require the grid to have a set height.

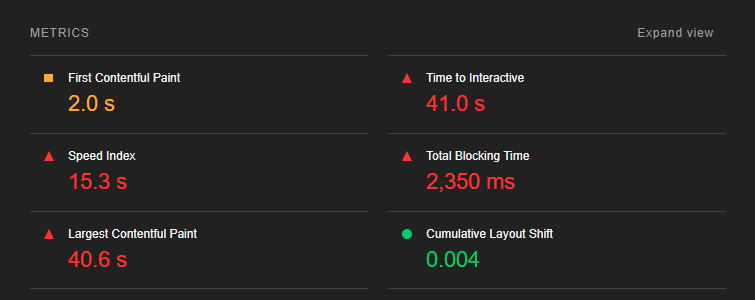
Lighthouse benchmark screen captures:

Loading 10 payments

Text

Description automatically generated

Loading 50 payments



Loading 100 payments

A screenshot of a computer

Description automatically generated with medium confidence

**Backend:**

Each time the “next page” button is click, we are hitting the same endpoint that returns all ERA payment information, basically the entire page is being reloaded instead of just the next payment list. Another unrelated call **loadEraCodesets**

With the ERA filters being applied to this page, Team RCM plans on breaking out the 9 queries that make up the **Claims\_LoadClaimPayment** sproc so these metrics only compare the current paging code.

Indicates the elapsed time to serialize data after the SP is called.

|  |  |
| --- | --- |
| **Records (paid claim)** | **Elapsed Time** |
| 1 | 1 ms |
| 50 | 1 ms |
| 100 | 3 ms |

Per Tiago previous spike: [Doc](https://centralreach365-my.sharepoint.com/:w:/g/personal/tiago_kocerka_centralreach_com/Ece636eUGItLsDVZgp6GSQoBpm6L53camzc2SBaJ2xJr1Q?e=AbJVaR)ument

*Based on ERAs from 2021, only about 1% of the ERAs contain 10 claims or less and only about 12% had 100 claims or less*

**Database**

The current sproc will take in a page size parameter so increasing the page size would not require db changes.

There are several select statements in **Claims\_LoadClaimPayment** that are performing table scans which requires SQL to touch each record in the table and looking at the execution plan several statements are giving warnings on excessive memory usage.

Query Statistics

Table

Description automatically generated

Example of SQL warning

Text

Description automatically generated

**Recommendations:**

On the front end we will need to use React-Window to reduce the number of DOM elements rendered at one time.

The stored proc should be replaced with separate calls made from the back end using an ORM which will allow the payments page to only request the information needed during paging instead of what is essentially a page refresh.