Performance Monitoring Header - UNO Capestone 2021

Introduction

Buildertrend would like to create a performance monitoring header to help collect and display key metrics to developers working on the solution.

There are many steps in a typical software development lifecycle, from development, code review, and testing. Any technical issues found during these steps require restarting the process. The sooner issues are found, the faster and cheaper it will be to fix those issues.

A performance header can help display key metrics to developers so that they are made aware of performance issues while they are working on work items. As a result, performance issues will be less likely to surface during testing, or worse, in production.

Minimum Requirements

- Developed in Typescript React and utilize Ant Design components wherever possible
    - Avoid using basic HTML components. Look through Ant Design’s catalog of components and use them instead wherever possible.
- The component is displayed as a sticky header at the top of a page.
- Track performance for things such as, but not limited to:
    - API Response Time
    - # of Database Calls
    - Database Time
- Create a simple website to test full-stack application to test the performance header on
    - Backend language can use whatever you want

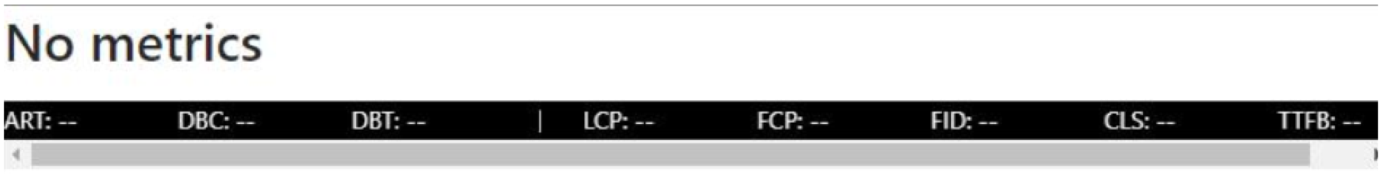
Other Goals

- Develop with extensibility in mind: how easily can you add or remove metrics from the header?
- Baseline monitoring:
    - Document API call performance to a database to create a historical record.
    - Use the historical records to create a “baseline” expectation of application and database performance.
    - Determine outlier values for performance to determine when an API call is far past the expected numbers.
- Support pages that make multiple API calls
    - If you load a page with multiple API calls, we care about the **sum** performance, not just the performance of the last API call. Consider grouping API calls made within 5 seconds of each other before displaying them on a header. Mousing over the statistic in the header could then reveal the individual API calls.
- Tracking other performance metrics
    - Front-end performance metrics? Web vitals?

Resources

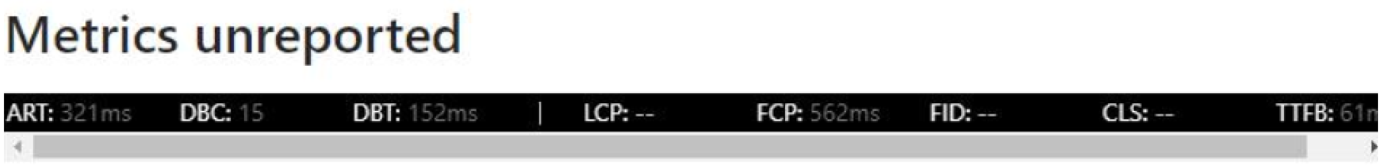
- React tutorial
    - https://www.youtube.com/watch?v=Ke90Tje7VS0
- Typescript “in 5 minutes”
    - https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-oop.html
- Ant design
    - Component catalog: https://ant.design/components/overview/
    - Installation instructions: https://ant.design/docs/react/introduce#Using-npm-or-yarn
        - Install using `npm`
- Utility for creating all of the boilerplate needed to create a new Typescript enabled react app
    - https://create-react-app.dev/docs/adding-typescript/

Screenshots



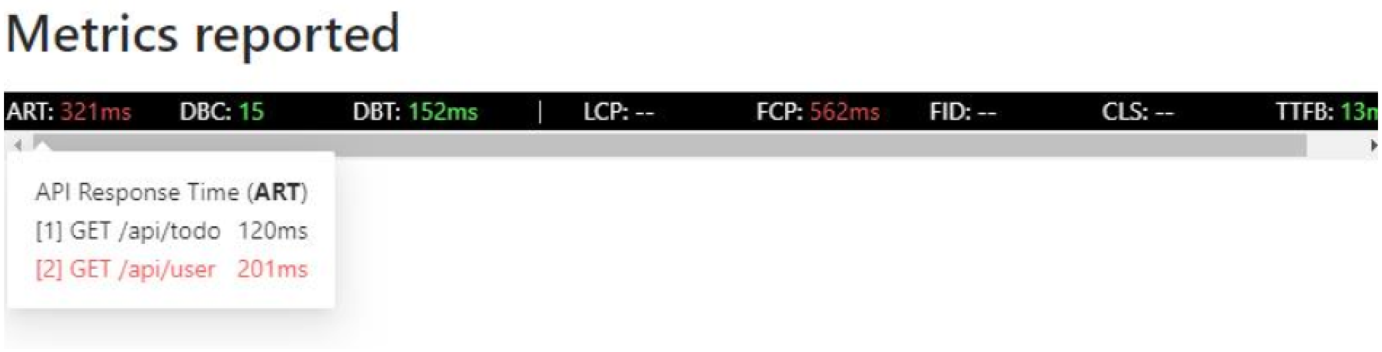
No metrics

ART: -- DBC: -- DBT: -- | LCP: -- FCP: -- FID: -- CLS: -- TTFB: --



Metrics unreported

ART: 321ms DBC: 15 DBT: 152ms | LCP: -- FCP: 562ms FID: -- CLS: -- TTFB: 61



Metrics reported

ART: 321ms DBC: 15 DBT: 152ms | LCP: -- FCP: 562ms FID: -- CLS: -- TTFB: 13

API Response Time (**ART**)
[1] GET /api/todo 120ms
[2] GET /api/user 201ms

Possible approach

- Endpoint: the React component tracks performance metrics, whenever it receives new values, waits 5 seconds for other values to come in, send them to a server as optional parameters for each statistic, server will return whether it's out of the nominal range
  - In other words, the "First contentful print" statistic could be reported way before we make any API calls but for performance reasons we should only "report" the metric to the backend/database after 5 seconds or so of "inactivity" where no new statistics are reported or changed.
- The # of DB calls made and the amount of time spent in the database for various API calls should be provided in the response header as it is currently in our Buildertrend solution. You can replicate this in your example application by adding fake/random values in every mock API response made on the backend.
- Response time of the request itself will have to be tracked manually.
  - Setup a service worker that intercepts all fetch requests in an environment tracks how long the fetch takes if it's an API. (broadcast API could possibly be used to communicate from the service worker to the React component)
- Libraries could be used to track front-end performance statistics (for example, Web vitals)
- Research and implement outlier detection algorithm to detect performance metrics that are outside of nominal ranges