Aalto-yliopisto
Aalto-universitetet
Aalto University

Master's Programme in CCIS

Performance Challenges with Data Visualizations in Browser Environment

Ravindu Rajatheva

Master's Thesis
2023

Aalto-yliopisto
Aalto-universitetet
Aalto University

Author Ravindu Rajatheva

Title of thesis Performance Challenges with Data Visualizations in Browser Environment

Programme Computer Science

Major Web Technology

Thesis supervisor Prof. Petri Vuorimaa

Thesis advisor Nokia, Pekka Suhonen

Collaborative partner Nokia, Matti Kosola

Date 10.03.2023 **Number of pages** 35+ 17 **Language** English

Abstract

Information exists in many forms, from text, to equations, videos, audio, and graphical mediums. With graphical or visual mediums, it is becoming easier to absorb information where the alternatives are textual descriptions. Graphs are important vehicles of transporting information. In order to create a good graph, certain attributes need to be taken into account, such as which variables are being displayed over which axis, visual elements, and their sizes are also important to consider. In modern times with the internet and the amount of data being generated, how can all this data be fitted into a single graph? That question is the motivation for this thesis.

Presenting large data in visualizations involves a great deal of thought, effort, and ingenuity on how to proceed with what information to convey. There are times when obtaining data for such visualization come with their own challenges. This thesis investigates the obstacles facing an internal tool within a company in regard to their data retrieval method. As well as the objective to research an efficient and easy-to-use method for presenting large data on a webpage.

Keywords ElasticSearch, Search-After, Canvas Element, Data Visualizations

Contents

Preface

I want to thank Professor Petri Vuorimaa and my colleagues Matti Kosola and Petri Rasimus.

Maunula, 10 March 2023
Ravindu Rajatheva

Abbreviations

Abbreviations

WWW	World Wide Web
W3C	World Wide Web Consortium
DBMS	Database Management System
RDBMS	Relational Database Management System
ES	Elastic Search
JSON	JavaScript
XML	Extensible Markup Language
SQL	Structured Query Language
API	Application Programming Interface
IOT	Internet of Things
M-SIM	Nokia Tool
GUI	Graphical User Interface
TA	Test Automation
REST API	Representational State Transfer Application Programming Interface
PIT	Point in Time
UI	User Interface
CSS	Cascading Style Sheets
V8	Google's open-source high performance JavaScript and WebAssembly engine
SVG	Scalable Vector Graphics
DOM	Document Object Model
HTML	HyperText Markup Language
JS	JavaScript
WebGL	Web Graphics Library
OpenGL	Open Graphics Library
D3	JavaScript Library for manipulation documents
RGBA	Red-Green-Blue Alpha
ID	Identity
SA	Search After
RXJS	Reactive Extensions for JavaScript

1 Introduction

From 1991 to now, the popularity and number of websites have experienced massive growth, affected both industries and in part even influenced society. The World Wide Web (WWW) started with only one website but now that number is over a billion according to one article [45]. The internet has introduced a new form of communication, new industries, transformed innovation and more. The growth of the WWW has brought along many adjustments or improvements to numerous processes over the past decades. Some of these adjustments include how media content is consumed. From watching programs on the television, listening to music on the radio – both can be achieved through online channels. This was one of the reasons that interested me to begin researching about web technologies since it has been and will continue to grow with time.

Furthermore, the WWW is still evolving with new emerging technologies either being implemented by the World Wide Web consortium (W3C) or companies developing their own techniques. These new methods can help solve some of the issues that currently occur when visualizing data. Some of these technologies that can help are canvas which was developed by Apple.

Data analysis is one field that has been grasped by the internet, specifically, when it comes to the size of new data. With more and more websites being present, approximately eleven users come online each second [45]. The amount of data generated continues to rapidly increase. To make sense of the data is itself challenging but presenting the data on browser environment brings forth additional difficulties. When it comes to displaying a large amount of data there are certain visual challenges that need to be overcome such as preventing visual clutter. These problems or restrictions require different strategies which include taking into account space or range one has to depict information. What is the end-user's goal for the data? Using these clues, it becomes feasible to draw coherent data visualizations.

It is very intriguing to explore the problems of visualizing data on a limited space, such as a webpage. This topic of exploring will be the main focus of this thesis as the problem gives birth to two questions, including the main question for this thesis. These questions are

1) What considerations should be accounted for when displaying large amount of data on a web page?
2) What method exists to present such data in a simple manner?

These questions will be covered and answered to extent sequentially in this thesis. The considerations take into account the end-user's perspective, questioning if all the data should be served to the user or only a portion. The

methods and frameworks for attaining data visualization are discussed at the end of the thesis.

This thesis will explore and describe the challenges and constraints of displaying a large amount of data within a webpage. Topics with visualization include screen space, visual clutter, as well as investigating methods to display meaningful data visualizations. Adding onto this topic, existing technologies that tackle and offer solutions to these problems will be reviewed. Furthermore, there is also the obstacle the company involved with this thesis, Nokia, is facing. The current implementation of their data fetching process within their internal tool, M-Sim, consumes an excessive number of resources leading to data retrieval failure. Upgrading the data retrieval process will be covered in the implementation section. Finally, in implementation section, a coding script will be explained in steps. These steps describe the process of producing interactive visuals, which can scale, on webpages while being relatively easy to implement.

2 Literature review

2.1 Background and Introduction

This thesis will introduce two challenges when presenting large amount of data on a web page. Large amount of data in this thesis refers to data consisting of hundreds of thousands to millions of datapoints or entries. The first problem is to retrieve data efficiently for the company given the development environment. The next challenge is to select suitable methods to visualize large, and varied data based on conditions given from the company. There may be many techniques to compress data into a presentable format, but it is still important to retain useful information which can be extracted from the visualization. The thesis will cover approaches and solutions to these questions to certain extent. Data retrieval issue facing the company in association of this thesis will be the first topic to be explained. Then the practical implementation will be discussed, including its benefits compared to previous implementation. Lastly, the final topic to be explored are methods of visualizing data and how to implement the canvas method. Based on the outcome, the one that is favored most by Nokia will be selected as the visualizing method.

2.2 Constraints

This thesis assumes the reader has previous knowledge or is aware on topics including Database Management Systems (DBMS), webpages, html documents, executing JavaScript code, and basic coding knowledge. This paper will focus on Elasticsearch (ES) with JavaScript Object Notation (JSON) data (used for analytics and monitoring log information). Other factors to pay attention to is the challenge of visualizing this large amount of data on client application or on browser. On occasion, the browser will not be able to handle large sets of data and halt the viewing application due to running out of memory. Visualizing this data will also have to overcome issues such as visual clutter and employ data transformations.

2.3 Storing Data

In database technologies, there are schema and schema-less storing specifications. Schema defines the types of database objects, relationships in data, and constraints of data fields [12]. The problem with using a schema is that it forces applications or technologies to consider “schema first and data later”, which can hinder the progress of growth or starting applications. Schema-less databases allow data to be stored with no pre-defined standard that are ready for use right away for application developers. Data used and stored in schema-less databases are referred to as semi-structured data and unstructured data. There is support for semi-structured data in Relational Database Management Systems (RDBMS) and DBMS. However, JSON

doesn't have as much support with Structured Query Language (SQL) as there is with XML (Extensible Markup Language) [9,10] in RDBMS. In RDBMS, the typical data format is stored in tables, where columns store attribute fields and rows represent an individual entry. These tables have attributes called primary keys and foreign keys that reveal relations between the data. XML data can be used as if it were relational data [11]. In contrast, to use JSON data in RDBMS, you would need a schema to exist in the raw data. That is not the cases most of the time. Instead, to use JSON data in RDBMS, there are usually work-arounds. With Oracle, JSON data joined with relational data to be compatible with its RDBMS [9].
With DBMS and schema-less databases such as NoSQL, some trends for data storage leans more towards JSON. JSON is natively supported in JavaScript, a language that has increased in popularity over recent years. Resulting in more adoption of JSON formatting in databases. Ultimately, these factors combine to favor schema-less database management systems since they support JSON format [1].

2.4 Schema-less

Schema or schema-less? Both these designs bring their own positives and negatives. On one hand, schema-less DBMS do not restrict developers to focus on a standard to store their data. Therefore, "work" can begin right away. Furthermore, companies or organizations possibly use more than one type of database technology – which gives another benefit to using schema-less database. The reasoning is that each RDBMS have their own query methods and language. This makes integrating, combining, and scraping data from different sources difficult. However, schema-less DBMS eases the process of common query language usage [1].
ES is a distributed DBMS. Built from Apache Lucene, it offers simple REST API's (Representational State Transfer, Application Programming Interface) for accessing documents, built for scaling, and compatible with many different data types [13]. ES is capable of using schema and working with JSON data. JSON format is associated with schema-less databases since JSON formatted data is loose in structure (semi-structured), so ES has the ability to be schema-less [3]. The main benefit of schema-less data is that it is suited for big data analytics, logging, and real-time data [2]. These factors are perfectly suited for Nokia's use case.

2.5 Schema

From ES' own blog, it is stated that ES uses a schema principle. The schema is in the form of a mapping that describes the fields in JSON document, and what their attributes are (data types). The JSON document contains fields

including document id, as well as text of the document itself storing information as key-value pairs.
Faceted search is also another benefit of schema-based RDBMS (Relational Database Management System). Faceted search or filtered search are queries made to obtain data with strong relation to other data in order to discover patterns or trends in the data.

2.6 Nokia Case

Nokia is a global company with customers all around the world. Nokia has an estimated 100.000 employees worldwide. Nokia produces a massive number of products where all go through extensive testing. Some of these testing or quality assurance is conducted through internal tools. These factors combine to create an environment where large amount of data is produced within Nokia, whether the data originates from IOT sensors or Base Stations or other devices there is a lot data from products that needs processing. As technology evolves and continues to grow, there are an increasing number of methods for data analysis. Such is the case with Nokia's case specific internal tool, which will be referred to as M-Sim or M-Sim tool in this thesis.
M-sim is focused on test automation. M-Sim uses simulated real networks to test variety of network components through several different types of tests. Tests include short bench, stability test runs, high-loading tests, and more. Analytical data is collected from these components to help understand and analyze behavior of networks. This tool is not involved with external customers such as everyday consumers, rather the end users are testers and other workers within Nokia. M-Sim tool's Graphical User Interface (GUI) maintains a dashboard displaying different test-runs and analysis results. M-Sim collects these data using test runners (Jenkins), data collectors, and analysis engines. This data is stored in ES but is accessed by ES' own API and served on M-Sim GUI.

2.7 Data Retrieval Challenge

To preface the one of the issues with M-Sim, consider the flow of end users when building a high-level structure for a commercial shopping website. This will include several components such as starting with a user interface to display the state of the service, the homepage of a shopping website, the store's inventory, and different events for the store. Furthermore, there is the backend logic to consider which processes user interaction information. Suppose a user who is clicking "add to cart" button on a product page. This information is read (product added to cart) and the database server of the store is updated with this new information. The database is connected by the backend, usually through an API (application programming interface). In

some circumstances, a customer may request to browse and look at all the products a shopping website has to offer. If the response for this request is sent from the server, it could include all products in the shop. If it is a store with many tens of thousands to millions of products, then the size of this data can cause the user's browser or client application to freeze. Or possibly to terminate. The solution to this problem is to segment the list of products and send them accordingly based on users' parameters. One such approach is called pagination.

Pagination within the context of web development can be understood as techniques to retrieve and send data requested by end-users in segments or in parts. Suppose an end-user is browsing an online store viewing the front or home page. There are categories listed on the page and the end-user wishes to see all products. Occasionally this list of products can range from hundreds to the thousands. Typically, this transaction would appear as:

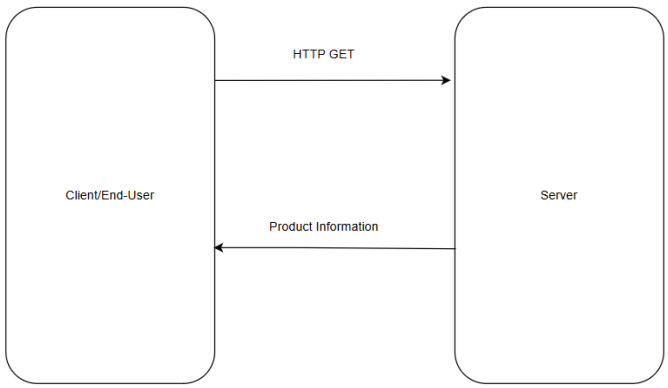


Figure 1 HTTP Get Request

However, the request above would likely freeze the customer's browser or application. Since the browser itself can only maintain a certain limit of memory to store all the content of the store's information. The solution to this large data load is to reduce the data by sending only relevant or appropriate data defined by the user. Other methods include to limit the range of information returned to user through techniques such as lazy loading with the help of browser API's, which will be discussed further in the paper.

Instead of the store's server responding to client request with a page containing thousands of products and potentially crashing the client's window. The server responds with a preset number of the most relevant products according to end-user criteria.

Relevance refers to products categorized and requested by the user. The products in the server are ranked based on the end-user's criteria and the highest ranked results are sent from database server to client side. This method of sending partial section of a collection of data is referred to as

pagination. There are several techniques for pagination, in this thesis offset-based and cursor based-pagination will be discussed. As well as ES' own scroll and search after mechanism will be discussed.
However, there are still many issues to pagination. Specifically, in offset-based pagination which is generally the same as reading the entire database until the desired results are found [4]. For instance, an end-user that is searching for results that are in the middle of a product list would be the same as the server going through the whole product list. This continues until the server finds the requested products and sends the information back to the client. This is quite taxing on computing resources, time, and the main difference is the data sent back to the end-user. Another efficient pagination method is cursor-based.
Rather than relying on client-side (user application, browser) cursor-based pagination uses the server to fetch the appropriate data or documents to send to the end-user [15]. This is less expensive on resources and uses less time. Cursors refer to the data that is being read or is pointed to at the moment.

2.8 Retrieving Data in Elastic Search

Similar to most online services, one would need a database to store all the service's information such as user logs, product listings, and more. A shopping website might store information on upcoming products, quantity of items in stock, and customer interaction. Transaction logs help to analyze trends in these interactions. In this scenario, patterns can be discovered by analyzing timeseries data of customer action. Imagine that there are developers working on this online store, where they plan to increase customer engagement. Perhaps by increasing customer retention. To achieve these goals, transaction information of customers interacting with the store is necessary. In a similar manner, developers of the M-Sim tool operate the same way, analyzing transaction information and building upon the findings. But these developers would need a dashboard to retrieve and visualize the stores transactions. However, the size of this information can be too large to load as a whole in browsers. If there is an application to view the timeseries data to see patterns or anomalies, it may be necessary to only load parts of the whole dataset. Occasionally, these sections of data might be in the middle of a larger dataset. Querying servers to get this data as a whole would require a lot of time and resources. Hence cursor-based pagination is used instead, in ES' case, search-after is the appropriate method.
When querying data in M-Sim, one would need to paginate data stored in ES. One method for pagination is the scroll method. By default, ES has a set limit for the number of results, called hits, for each request. The limit for responses is set at 10.000. In order to get more hits, the scroll method or search after method should be used.

Another challenge when someone wishes to analyze data in M-Sim is that the data might be modified by external entities (analysis engine). The problem with reading data entries is consistency in the data since there might be a lot of writing to the data that will be fetched by queries. In M-Sim environment, there are many TA (Test Automation) system collectors and analysis engine which writes data to ES server.

2.8.1 How Data is Served on M-Sim Tool

There are several operations that can fetch information or documents from the database server to serve on the endpoint (target website). The GUI is the visual component, but the information served on the GUI is gathered from an API. ES has their own SQL REST API (Representational State Transfer Application Programming Interface) which is exposed to the web service. End users on websites will select data they wish to analyze leading to the API to create aggregations to fetch from database. When the transaction is complete the API sends the data to the GUI where it will be rendered on page.

2.8.2 Elastic Scroll

The first pagination process present in M-Sim is the scroll method. Which calls or pings the ES API where the response would be scroll context id. Context refers to the state of the index of data being viewed and subsequent changes to the documents are ignored, thus consistency is preserved, and readers don't have to worry about writes to the data. The issue with this context is that it reserves memory in ES server. This method can keep reserving memory until there is no space left which can halt M-Sim tool. Therefore, it is recommended for GUI services to limit the number of scroll context to 500 [16].

2.8.3 Elastic Search After

The newer method to fetch large collection of documents (greater than 10.000) is called "search after". Unlike scroll method which is outdated [14], this method uses a "point-in-time" id, or PIT id, which is more lightweight in comparison to scroll context. Using the id, the contents of a data index is the same as it was when the point-in-time ID was created regardless of whether other users were writing or modifying the data being read using PIT id, similar to scroll context. This is for consistency and robustness, so that if other parts of the index were modified, the search after call would still work. In comparison to scroll, this method is lightweight and is not limited.

2.9 Data Visualization Challenge

One of the general issues with visualizing large sample of data comes from the size of data. It is not feasible to understand granular information or be able to recognize patterns in data if there are millions, billions, or more datapoints present in one graph. It is not sufficient to extract useful information from such a graph. Furthermore, it is also unlikely there would be a graph large enough on a webpage to render a billion or more datapoints. Therefore, there should be techniques performed on the data and graph in order to make a graph easy to comprehend and to convey information. But how does one decrease visual clutter, what areas of a graph should one focus on? Is it appropriate to compress multiple datapoints into a singular point? Perhaps data shading is a better method to display trends in the data. This topic will be discussed further in the literature section.

2.9.1 Limitations of Visualizations and Screen Size

The main problems with visualizing large amount of data comes with, the size of the data. Taking into account the size of a computer screen there is only so much space to draw data points on a graph. Another point to consider whether users who are analyzing the data may have different screen sizes and may prefer to not have their whole computer screen be occupied with a data visualization. Workarounds to consider when presenting large amount of data can be grouped as limiting the scope of the data and using different techniques for data rendering.

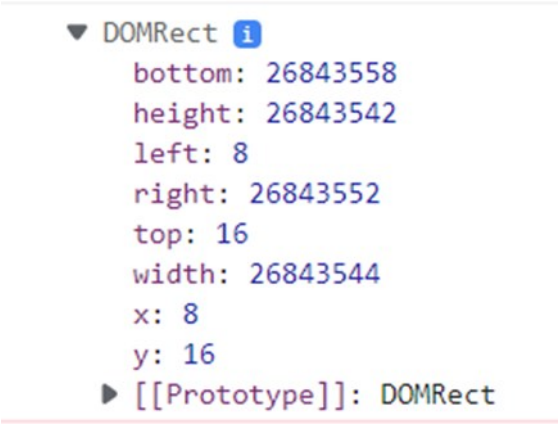
Under the category of limiting data and compression are data transformations. Data transformations refer to the manipulation of data – commonly known techniques include mean squared distance, normalizing the data, categorization, and dimensionality reduction. Dimensionality reduction is important approach for when one's data contains too many features. A feature refers to different attributes of data. Consider a dataset containing high-school students. Each student is an entry of data, or the "y" variable, however, each student can have characteristics such as height, gender, and major ("x" variables). It is commonplace for datasets to contain thousands or more features. Reducing the data helps to improve the efficiency of analysis as well as its accuracy. Another problem with high number of dimensions is that they can degrade the performance of machine learning algorithms and pattern recognition applications. In order to select the most useful features during data analysis, it is important to select a suitable representation of the high dimensional data in the preprocessing step.

Additionally, there is also feature transformation which is the process of transforming the original list of features to a more compact set of dimensions. This topic can be divided into feature extraction and feature generation. The former is the practice of combining features into a single feature thus creating new features (some mapping is required for this selection). Feature generation is defined as ways to discover missing or new information

between features from the original data set [46]. Ultimately, there is also feature selection which refers to choosing the features that are the richest in information.
Another method for limiting the scope is to actually limit the data one can see. If it is not possible to drop any features from a dataset, and neither are transformations of data allowed, then there is still the option to focus on specified range of data. This perspective is more on web User Interface (UI) perspective than it is for data analytics. With modern frontend frameworks such as React or Angular there are many technologies, which can detect where a user is currently located on the browser window. Therefore, it is possible to only load a portion of a dataset to an end user. Relaying the appropriate range according to user location makes it possible to offer the user a seamless experience for viewing data.

2.9.2 Limitations on Web Side

Rendering large datasets using tables can already be an issue. Most browsers have a limit for their maximum pixel count or threshold for an element. Using CSS styles, one can create an element with a width of 100-trillion-pixel size. However, when checking the actual size of the element using "DOMrect" property, it is around 27-million-pixel size.




```
<body>
    <p style="width: 100000000000000px; height: 100000000000000px" id="test">Title</p>
</body>

<script>
    const item = document.getElementById("test")
    console.log(item.getBoundingClientRect())
</script>
```

Furthermore, computer screens already enforce a limit to how many pixels can be rendered on the screen. Especially, when presenting dense datasets on the on-browser windows, it may not be safe to assume end-users possess large displays to be able to view a complete dataset. Since the web is

ubiquitous, the approach should be a general approach to display data such that any user can gain valuable information.
End-users or client-side are only able to download certain limit of data (in chrome's V8 engine the maximum size is 512 megabytes on 32-bit systems and 1.4 gigabytes on 64-bit systems). Recently, that limit has been pushed up to 4 gigabytes [5]. With trends of big data only getting bigger this memory limit will be reached as well and other alternatives are needed.

2.10 Web Standards

Normally, when thinking of images, the format is usually raster bitmap images. Raster images consists of pixels in a grid formation to create an image. Color values and tonal information are combined and help to produce an image. The limitation with this format of images are their pixel counts, meaning they are resolution dependent. Suppose if one wants to enlarge a raster image without increasing the resolution, the number of pixels in the image will stay the same. The pixels will get larger, and resolution will degrade [34]. Therefore, the more pixels a raster image has, the higher quality the image will be. On the other hand, there is SVG, scalable vector graphics.

2.10.1 SVG

Scalable Vector Graphics or SVG is a web standard. With SVG you can draw or visualize three types of graphics, which include vector graphic shapes, images, and text. SVG itself is a language for describing two-dimensional graphics in XML. Graphical objects in SVG can be grouped, transformed, styled, and composited into previously rendered objects. Several features of SVG for these objects are clipping paths, filter effects, nested transformations, and more [33].
The main benefit of SVG is that it can be both interactive and dynamic. SVG can be animated, this can be achieved by embedding SVG animation elements in the SVG content. The animation can be then triggered explicitly, or through scripting. Event handlers on the DOM can be assigned to SVG elements. SVG supports properties of CSS and is compatible with HTML [33]. Final content of the SVG result is drawn on the SVG canvas.
In regards, to image quality SVG, unlike with raster images, is able to maintain its resolution independent of its viewport size using matrix transformations. Transformation matrices define the process of mapping one coordinate system to another. SVG holds a user coordinate system to maintain the locations and distances of the user's current canvas. Instead of tracking numerous pixels as raster images do, SVG keeps track of points as well as the equation for the lines which connects those points. To reiterate, SVG maintains the transformations of points. This frees up the resolution boundaries of raster images and allows SVG images to be re-sized indefinitely.

However, there are limitations for SVG. A paper by Jen-Chang Liu [35] focused on converting comic images with text to SVG. Some of the problems highlighted in this paper discusses the large file sizes created by SVG images. When converting raster images to SVG, the file size would be too large. This is due to the information of points and paths present in SVG. Another limitation of SVG is that the "base" image to convert, or image to produce must be simple. For graphs, it can depend on whether SVG is a suitable candidate, since it can scale as much as one wants with no loss of information or quality. Since SVG can be animated, the graph can also be interactive. But as previously mentioned, large file sizes and large number of datapoints in a graph can hinder the benefits of SVG. Large number of datapoints require more points and line transformation data which will increase file size of a graph. For this reason, it might not be suitable for visualizing large datasets.

2.10.2 Canvas

Canvas was first introduced by Apple in 2004 in Webkit [40]. It was designed for rendering graphics and animations on the client rather than the server. This allows for bottlenecks of the server and bandwidth restrictions to be prevented [41]. It was later standardized and supported in the HMTL5 standard [37]. It is a DOM element which can render two-dimensional graphics on a defined bitmap. The name canvas refers to the container which will render the bitmap. Canvas graphics are created using two-dimensional coordinate system and it is the only context available on browsers supporting canvas. Canvas is a commonly used low-level technique for displaying graphics in JS libraries due to its high performance [38]. However, after the visualization is drawn on canvas, information on the drawn objects is not stored since canvas is a single DOM element. Immediate mode refers to redrawing the canvas whenever there are changes made. An advantage of this is the ease of applying changes to canvas through global properties [36].
In regard to performance, canvas only has instructions on what to draw on any single frame. The rendering occurs in the client side so graphics heavy pages render faster [41]. This makes canvas performant, but due to canvas not retaining information on the graphics within the container, canvas by itself is not good for interactive visualization if one wants to gather data values by clicking on the canvas.
The HTML canvas object itself has many properties and methods that can be accessed by JS. Main properties define the size of the canvas width, and height. Next are the available methods, which are used to create visualizations. These methods include drawing shapes such as rectangles, lines, and as well as bezier curves. Then are transformations, which can scale, rotate, and translate. One can also create a clipping region with canvas. Canvas also comes with several properties to manipulate the context such as "strokeStyle", "fillStyle", and more. The size of a canvas grid can also be accessed

through CSS styles. However, it is not resizing which occurs when increasing or decreasing the size of the canvas, rather resampling is done instead [36]. Canvas applications are also self-contained since the canvas grid is displayed in a defined region of the webpage. Therefore, the canvas should not interfere with other elements occupying the same page [36]. A paper by Daniel E. et al from Tufts University compared several data visualization techniques. The findings showed that a combination of WebGL with canvas was the fastest. However, in second place was canvas combined with SVG. Due to its low-level architecture and optimizations, canvas is a good approach for visualizing large datasets.

2.10.3 WebGL

WebGL is a 3D drawing API for HTML canvas that originated from Mozilla engineer Vladimir Vukicevic. Vukicevic wanted to create a 3D drawing with html canvas context, by 2007 there were implementations of canvas 3D for Mozilla and Opera. Finally, in 2009 Vukicevic was joined with engineers from various technology companies to create the WebGL working group [43]. WebGL is a low-level API that is based on OpenGL. It is cross-platform and is exposed to ECMAScript via HTMl5 canvas [42]. WebGL lack few high-level constructs, but there are open-source JS toolkits that provide high-level access to the API to complement WebGL, so it feels closer to a traditional drawing library. [43].
Before the introduction of WebGL, developers had to rely on plugins or native applications which ask users for their permission to download and install custom software. For the goal of delivering a hardware-accelerated experience. It must be stated that WebGL is not in the HTML5 specification, however, it is shipped with most browsers that support HTML5 [43]. The major difference between WebGL and Canvas are the dimensions. 3D rendering is possible with WebGL while canvas is meant for 2D drawings. But both APIs are accessible through JS API calls. WebGL context is accessed through canvas element using special drawing context that is specific to WebGL.
Although WebGL is fast and performant, there are other aspects which might dissuade it as an alternative for data visualization. WebGL itself uses JS, but to use it effectively the developer needs to have strong mathematical skills as well as experience with OpenGL and shader language.

2.10.4 D3.js

D3 is a popular JS library for building interactive data visualizations that output the final graph or chart into SVG format. D3 uses SVG, HTML, and CSS to apply data transformations directly onto the DOM object. Therefore, there is no separate charting/graphing software, and it also means no propriety visualization/representation. Since visualizations are created using web

standards (HTML, CSS, and SVG) [30]. D3 uses a list of commands or keywords that are similar to jQuery's. Although D3 is extremely fast, there are limitations as there are with any technology. To use D3 effectively can be a steep learning curve, while other data visualization technologies can even be drag-and-drop. Meanwhile to use D3 one must be familiar with JS, HTML, CSS, and in general coordinate geometry.
A paper by Lehka et al. found that D3 was not able to visualize an entire dataset containing an approximate of 6 million records [31]. Though the paper is from 2016 and there have been progress made to improve the performance and speed of D3, an important issue to be taken into account is that visualizations are in the form of SVG. SVG visualizations in D3 are bind onto DOM objects. Having too many DOM elements can degrade the performance of webpages. In general, to optimize a webpage it is recommended to limit the number of elements in the DOM. Recommendations include to limit DOM size to 1.500 elements, a tree depth of less than 32 levels deep, and fewer than 60 child/parent elements [32]. Going past this limit can cause the UI of a webpage to be slowed down since more elements need to be rendered and more resources are consumed in the process.
This constraint of on the number of DOM elements as well as the learning curve show that D3.js might not be the best data visualization JS library in regard to visualizing large datasets. While D3 provides great performance, support, and more for interactive data visualizations. D3 utilizing SVG to visualize and render datasets with millions and more datapoints is not yet present or tedious to do so. However, there exist numerous alternatives to use for data visualizations which do not have the shortcomings of D3.js. Some of these alternatives include Zing Charts, Apache Echarts, or Chartjs. The technologies used in these libraries include a mix of WebGL, HTML canvas, and SVG.

2.10.5 Apache Echarts

Apache E-Charts is an open-source JavaScript visualization tool. This library supports numerous chart visualizations including bar, line, scatter plots, candle-stick, and several more. Charts are rendered using canvas, SVG, and VML (Vector Markup Language). E-charts can also provide three dimensional visualizations on top of two-dimensional charts. E-charts has stated to be able to display tens of millions data on web [44]. On the home website there are plenty of examples, guides, and walkthroughs to help users get started right away. E-charts also has a large and active community, which makes E-charts a perfect candidate for visualizing data in the M-Sim tool.

2.11 Proposals/Possible Solutions

During the exploration of the issues faced by the team at Nokia in regard to the data visualizing problem, there are frequent topics that pop up. These include performance, scope of data, and the technique for rendering/drawing method of the data. These are key steps to consider before implementing an appropriate data visualization. Although, M-sim tool enforce some restrictions, solutions outside this boundary will be explored as well.

2.11.1 Performance

With the use of web-workers it is possible to create separate threads in JS that manage other work while the main thread handles the rendering of a web page. The requirement for web workers is more of a consequence to the structure of working with browsers and due to JS' single threaded nature. In JavaScript there is an event loop which helps with reserving processes taken from the main thread, to execute until after the remaining functions or processes in the main thread are executed. Few reasons for the prioritization of certain functions are due to asynchronous flow of execution when communicating with external data sources.
Asynchronous programming is necessary in the user experience of web pages. If JS executes network requests synchronously it is possible that when an end-user checks the live weather in their hometown. They may first proceed to this weather website, where a container/html element would display the weather information such as temperature, humidity, and other relevant statistics. The live information has to be updated periodically, once this occurs, the end-user would be unable to interact with this weather website until after the weather statistics are loaded once again.
This blocking would impede development since many services depend on communicating with an external data source. If this communication were to occur synchronously it would lead to poorer user experience as webpages are blocked or loading before information can be served to the end-user as well as interactivity. Asynchronous programming allows JS to execute functions even though there might be some waiting for responses from external data sources. This execution flow is similar to what can be achieved with web workers.
One problem with visualizing large data sets originates from the volume of data. There are cases when end-users consider viewing data as well as observing data through different data transformations. It could be that users wish to see averages, normalizing, or other functions transforming a dataset. If this data is presented on a webpage using JS it could be that the main thread is already consuming a lot of resources, the data visualization webpage could be slow or unresponsive. This would be an ideal situation for web workers, which can be used to calculate the new, transformed data, message the values to the main thread which only needs to handle the rendering. This approach facilitates JS to be more performant when producing data

visualization since this would take less time, while providing the same functionality. However, limitations on the web workers are that they themselves cannot affect the DOM. This means that web workers cannot affect page structure directly, they can only communicate and pass values to the main thread.

2.11.2 Scope of the Data

Another aspect to consider which has been mentioned several times in this thesis already, is the scope of the data. Scope of the data refers to the size of data that will be visible to end-users. Due to the restrictions of screen size, displaying a whole dataset within an html container or portion of a webpage could lead to visual clutter. To avoid this issue, data reduction techniques can be processed on the data such as dimensionality reduction, filtering, and other transformations. However, an alternative is to visualize the data without changing the dataset, but instead to only show a portion of the data. Within browsers and with help from JQuery it is easy for developers to track end-user's mouse pointer position through DOM API's such as "mouse move", "event.pageX", and "event.pageY" [47, 48]. The mouse move API will trigger when a mouse pointer/cursor is within a html container such as a div. Once this event is fired then other API's can be used in conjunction such as the event.page which will return the X and Y position of the cursor within the html container [48].
Acquiring the cursor positions is one part of learning a user's location in a page. In addition to tracking users' cursor positions, it is also useful to track mouse wheel events emitted by end-users. Mouse wheel or scroll button is another tool which can help to move around in a webpage. Using the "mousewheel" API developers can check when and whether end-users' have used the scroll wheel [49]. Cursor location and scroll button are information which can be referenced to understand where an end-user is currently, as well as their behavior. Behavior relates to what actions the end-user is performing such as, is the user scrolling down or up, are they dragging the scroll bar horizontally or vertically? By placing your data visualization in a container, it is possible to check where an end-user is currently positioned in the container. By learning this location, the data loaded into this container can be a set size which occupies the container's size and not the whole dataset. This container would be interactive as well so that users can drag or scroll within the container. Such an example of data visualization for a website would be similar to the image below.

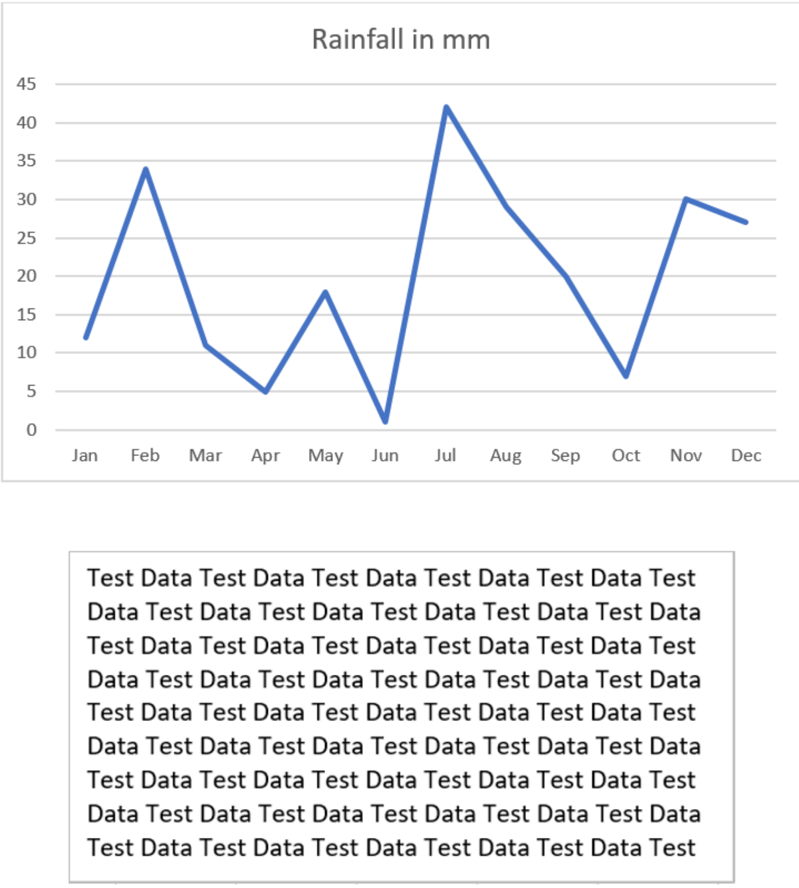


Figure 2 Time Series Data Example

A timeseries is as its name implies data over time such as weather data over multiple years. Timeseries data can be, for example, rainfall throughout the year in a region, snowfall height, as well as temperature over many decades. Suppose we have a dataset, which contained the rainfall (measured in millimeters) per month for numerous decades. To compress this information and deliver it to the user is the challenge to solve. Data reduction, transformation, letting the users' download the data or only provide the most valuable information such as peaks in temperature over the years. But, with the limited scope method, it is possible to load a portion of the dataset so that users can see and scroll through the data. The caveat of this method is that to find points of interest in the data users would have to traverse through the dataset. Furthermore, the context of the data can be misrepresentative since users can see peaks or drops in data, but these are only relative to the data being visualized and not the whole data. However, it is one viable solution to loading large data sets.

2.12 Problems Presenting Data on Browser

M-Sim is a testing tool which contains logs data, analytical data – both constituting vast number of dimensions of data. However, what is an efficient method to display this rich information that is digestible for the users? There might be many answers to this question but another factor to consider is that M-Sim is a tool that is occupied in the browser. The limitations of browser platform will be discussed in this chapter as well as workarounds to some restrictions that is brought with displaying data on browser.
Tables and graphs will be the focus for displaying data. Since they are the most common methods within M-Sim tool for data visualization. The tables in M-Sim containing columns and rows are those that contain more "tall" data, which means there are more rows in proportion to number of columns [22].

With browser there are two perspectives to pay attention to when rendering large amount of data. First consideration is loading the data in memory. In this context, large data can be referred to an amount that would overload the browser's memory causing for browser to "terminate". Additionally, another definition includes data that must be stored in a cluster of nodes in a server. Second part is more involved with JavaScript's single threaded nature. When JS executes a network request it is typically done in an asynchronous fashion. One key issue that takes place before rendering big data in a browser window is loading the data. Likely, the size of data will be larger than the memory available to the browser. To circumvent this obstacle, one can split the data and serve the relevant information on webpage according to user requirements.

2.12.1 Lazy Initialization

One of these methods for serving data is called "Lazy loading." Lazy loading or rather lazy initialization [23] is to defer the initialization of an object until the object is first used. This is to prevent loading unnecessary data/objects thereby consuming resources. This method can be used to optimize webpages. Use cases include users on shopping sites who don't navigate to a store page with a lot of browser events, such as a page displaying all the items on clearance. Therefore, there is no need to load this page and other expensive processes.

2.12.2 Single Threaded

Another factor to consider as mentioned before is that JavaScript is a single threaded language [24]. This refers to JavaScript executing functions one at a time synchronously. Assuming JS script called "foo" which consists of three functions: A, B, and C in which all are called sequentially. The script will first execute function A then B and finally, C. When running program "foo", the

flow of execution will be A, B, and C. However, problems occur when a webpage/website contain a script which takes a long time to load or is in the process of execution. Such cases where it is not known how long some functions will execute for on a webpage can halt and make a webpage unresponsive to user interactions.

2.12.3 Web Workers

Web workers is an API for running scripts in the background threads [25]. Worker is an object that runs a referenced JS file – which is the target file for the code a user wishes to execute. As mentioned before when executing expensive processing, which can block the main thread of execution, workers can be used to run these processes in the background to avoid the UI (in a webpage) from slowing down or being blocked. There are some limitations present with using a web worker, direct DOM manipulation is not supported, and other restrictions prevents running any form of code within workers [27]. Communication between the "main" script and "worker" scripts are handled through using the "postMessage()" method and responses are managed with the "onmessage" event handler .
Essentially, when there are expensive processes in a main script, they can be offloaded to a worker script. This functionality helps circumvent the flow of execution as previously mentioned.

2.12.4 Current View in DOM

Using JS frameworks to create dynamic single page applications, SPA's, there are features such as virtual DOM in react. Virtual DOM is a concept where the virtual representation of a page's UI is stored in memory and later syncs with the DOM. In react, libraries such as "ReactDOM" facilitate this syncing procedure, it is called reconciliation [28]. This feature helps developers to clear overhead of what events to handle, attributes to manipulate and focus on what the DOM should render at a given time. Essentially, developers can tell what the current DOM state should render.
Angular framework achieves a similar effect of virtual DOM through the combination of change detection and zones to update the DOM. Angular executes change detection whenever there are changes to data. Zone refers to preserving execution context across asynchronous tasks so that values, or data remains consistent [29]. Change detection and virtual DOM makes it possible to limit a webpage to only what a user requires. This minimizes page sizes, memory consumption, rendering time, and further reduces bloat on webpages.
Virtual DOM and change detection are viable solutions to reducing the scope of data needed to be loaded on a webpage. Suppose end-users are scrolling through a large dataset. One factor to increase responsiveness of the webpage

is to only load the data that is currently being rendered on the end-user's screen.
The features mentioned in this chapter includes lazy loading, web workers, and the concept of current view in DOM. These are several solutions which can be used in combination for different use cases to create responsive webpages that need to display a large number of data, complex data visualizations, and more scenarios where large amount of data needs to be viewed. Currently, M-Sim tool uses some of these approaches in several webpages.

3 Implementation

In this section, two topics that will be covered including the upgraded data fetching process in the M-Sim tool. While the second topic will provide a simple process of using the HTML canvas to facilitate interactive visualizations. Other methods for data visualization on the web use SVG formats, while others use WebGL. Due to certain restrictions from both of these technologies they will not be pursued further for use in M-sim.

To use a canvas element in a webpage simply use the "<canvas>" tag in an html document. Canvas elements on default have a width of 300 pixels and height of 150 pixels [51]. The canvas itself is simply a drawing surface, which is used to render context – which are used to manipulate and create the content within the canvas. A canvas is initially empty and requires a script to be able to render content. This canvas surface can be used to generate graphs, charts, and other data visualizations since it uses its own cartesian plane for locating pixels within the grid. The origin of the canvas grid begins in the top left section [52].

The "getContext" command is used to get the rendering context as well as associated drawing functions. To create two dimensional renders, one would use "getContext(2d)" paired with the associated canvas to instruct the canvas to render a two-dimensional drawing. To be able to draw shapes in canvas element, the "fillRect" command is used which requires four parameters (x coordinate, y coordinate, width, height). "fillRect" is used to draw rectangular regions on the canvas drawing surface. Similarly, the function "strokeRect" can be used to draw outlines, and lastly there is "clearRect" to remove the drawn shapes. It is also possible to create a path in canvas, essentially a series of points connected by a line. Nevertheless, in this thesis the focus will be on the "fillRect" command.

With the purpose of displaying interactive canvas elements, a simple bar chart will be used as an example. There are plenty of resources and libraries online which simplifies the process of rendering canvas elements without needing to use "fillRect" commands. Instead, one would just type data values and data indices to render graphs or charts. For this example, however, a skeleton code is used from Tanu N Prabhu's article on plotting bar charts with JS and canvas [53]. Using the column values from the code, the bar chart below is rendered.

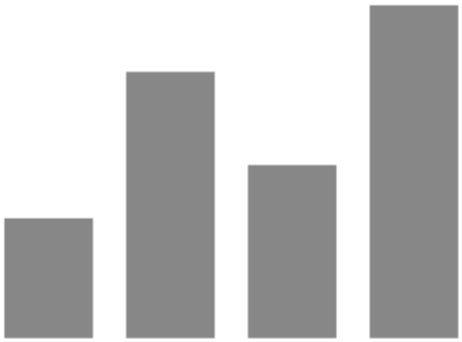


Figure 3 Bar Chart made in Canvas

“GetImage” data is the attribute, which enables the html canvas element to have interactivity. This command returns color and other detailed information on a specific pixel within the canvas grid. Therefore, it is possible to attach an event listener to a canvas, meaning that wherever the target canvas is clicked, information about the canvas on that specific location can be read and stored. If the canvas element above has an event listener and a user clicks on one of the grey columns, RGBA information from that click would display the color as grey or 128,128,128,1 (RGBA values). This makes canvas a powerful tool since it allows developers to utilize the pixel information. The canvas color information is bidirectional – meaning that pixel value information is given through a numerical array, and it is possible to modify the same array and apply the changes back to the canvas. This allows canvas to easily control and change displaying information on canvas. “Getimage” specifically returns the current state of the canvas as a list of integers. The list of integers relating to the number of pixels in each row of data, number of pixels in each column of data, and the actual red-green-blue [55] as well as the alpha (RGBA) values of the pixels.
However, manipulating the canvas is not needed in this section. Only retrieving the color information will be relevant in facilitating canvas interactivity. The parameters in the “getimage” function allow to specify a rectangular region so it is possible to read the color values of one specific pixel. To locate the component values of any pixel at given coordinates x and y simply use the formula below:

- Red component: ((width * y) + x) * 4
- Green component: ((width * y) + x) * 4 + 1
- Blue component: ((width * y) + x) * 4 + 2
- Alpha component: ((width * y) + x) * 4 + 3

The formula is given in the textbook “Using the HTML5 Canvas API”, chapter 2 page 58. To replicate the effect canvas interactivity, the first step is to show a bar graph where each column is a unique color. In summary, a graph as pictured below.

Figure 4Multiple Column Bar Chart

There are 5 columns present in the graph with their own unique colors. The colors represent individual indexes, columns or datapoints. But in M-Sim environment, datapoints or columns in graphs will have only have one color. Additionally, for data visualizations to be used in M-Sim tool they should be capable of interaction. For instance, when a user hovers their cursor on top off or clicks column 3, they will receive an alert or text block displaying the value in column 3.

To keep the example, brief and simple, a bar chart format will be used for demonstration. When there are a small number of elements (tens to thousands of columns), one canvas should be sufficient. But when data reaches sizes of thousands to million datapoints, one canvas element would be a bottleneck and rendering would take too much time or browser runs out of memory, or the webpage where the canvas occupies can be slow to respond – all of which are unfavorable conditions. Although, there is a limitation on the number of datapoints possible in a single canvas element since there is a set pixel size which is possible to display in one canvas element. That size limit, however, with RGBA values is 16777216 color values excluding the alpha parameter, which will further increase the size. On the other hand, there

is the restriction with these numbers since most end-user's screens will not be able to handle such a large number of pixels.

3.1 Interactive Behaviour

To create interactive canvas elements for use in M-Sim tool, two canvas elements will be used. One for the actual display while the other will act as a dictionary. Before the dictionary can be created, we used the unique color values to generate a bar chart with 4 different colors on a webpage, similar to the multiple column bar chart.

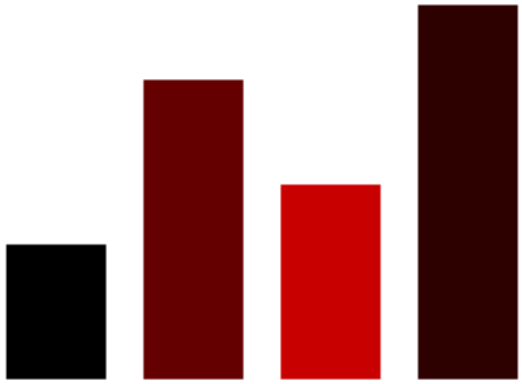


Figure 5 Canvas Bar Chart with Unique Colors

The unique colors will be the key to retrieving which column was clicked, in essence this can translate to a user clicking a specific point or pixel in the canvas and they will know the values thus we are able to gain interactivity through html canvas. These unique colors will be used on the hidden, invisible canvas as their column's colors. The unique colors act as distinct keys to find the values of columns in the visible canvas. The obstacle with this process is connecting the two canvases.
By using a dictionary object to store indexes with unique colors, it is possible to take advantage of the bidirectional pixel information when using the "getImageData" command. The main reason for using two canvas elements is due to the nature of data visualizations. Having too many colors can hinder rather than aid processing visual information. Therefore, the hidden canvas – the canvas with unique colors has their display attribute set to "none" so it is invisible on a webpage. While the uniformly colored, or "main" canvas will be the only rendered canvas. The goal to achieve with two canvases is the process as presented below:

Figure 6 User Journey

If a user clicks or hovers their cursor on a column or point, they will be able to retrieve information on the data. Either through a tooltip notification, pop-up, or other methods. Although, the process seems simple, another canvas is needed to process the cursor coordinates and retrieve the correct values at the cursor location.

To achieve this interactivity requires the use of a dictionary object to connect the visible canvas and the hidden canvas. This dictionary object, as its name implies, functions as a dictionary where column entries or indexes point to unique colors. These are key-value pairs, where the unique colors point to the column information.

The dictionary object stores the unique colors as keys while the associated column/data information is in the value field.

```
{
"uniqueColor1": {column 1 information},
"uniqueColor2": {column 2 information},
"uniqueColor3": {column 3 information},
}
```

In the user journey, the missing step was the hidden canvas. The hidden canvas is the same size as the main or displayed canvas that is shown to the user. With the only difference being that the hidden canvas is invisible. The hidden canvas possesses the unique colors for each column value. Using the mouse coordinates gathered from the click location on the main canvas, these coordinates can be translated onto the hidden canvas. These coordinates will be used as parameters in the "getImageData" function to retrieve the unique color at the cursor location. From then on, this color can be used to find the corresponding column information by using the using the dictionary object.

The missing sequence in the Figure, "User Journey" is that cursor coordinates are grabbed from the click event. The whole sequence is presented below:

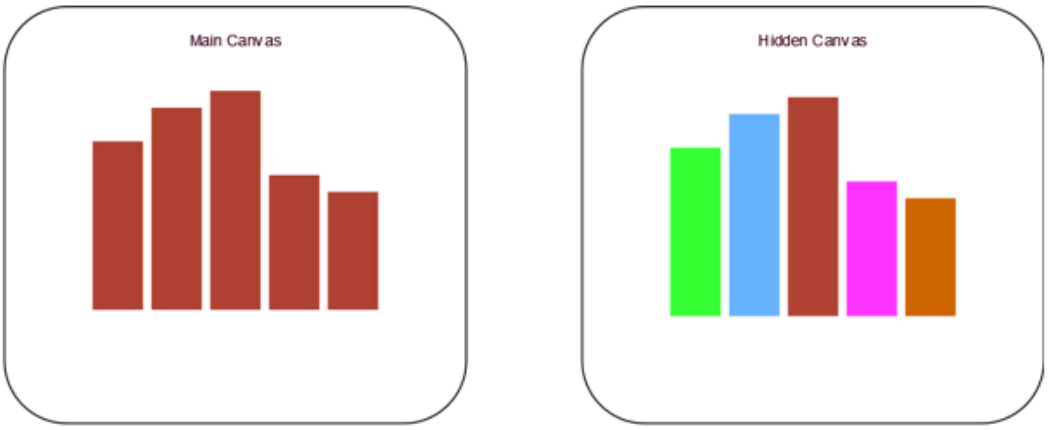


Figure 7 Render Two Canvases

The first step is to create two canvases but to render only one, the main canvas. The hidden canvas will generate unique colors for each column entry.

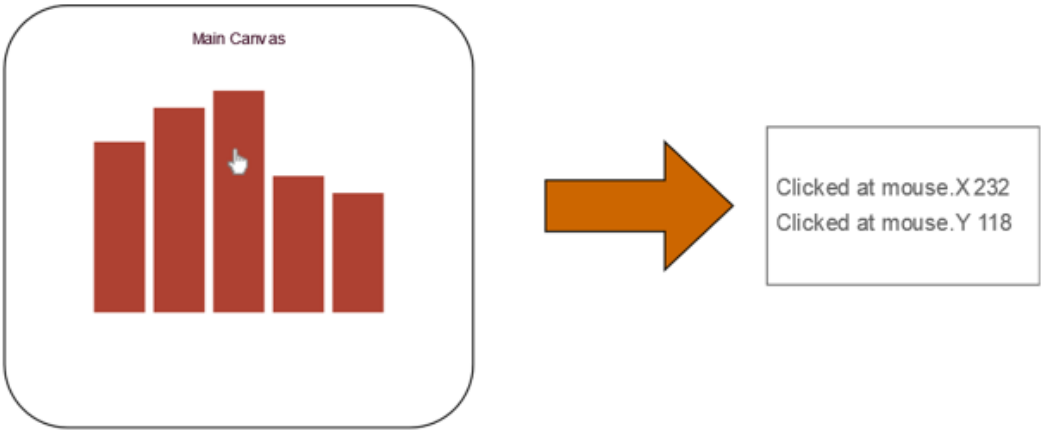


Figure 8 Main Canvas Click Event

When a user clicks on the main canvas the coordinates will be stored for later use on the hidden canvas.

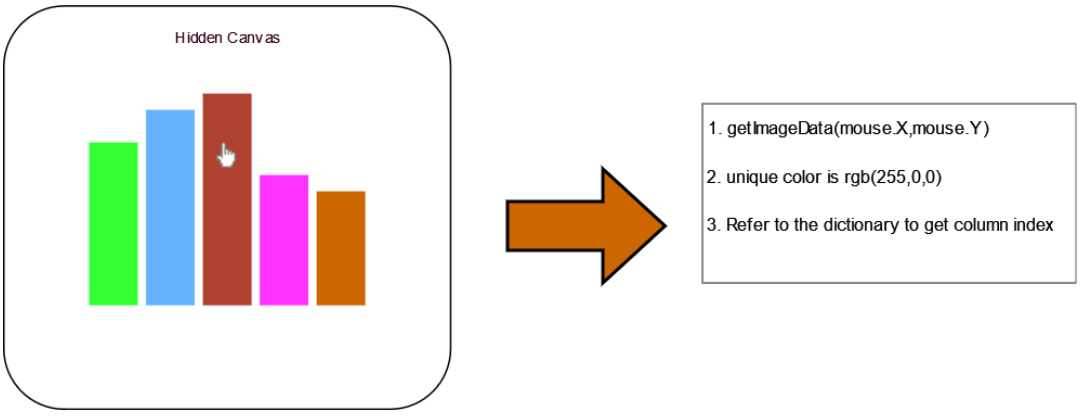


Figure 9 Hidden Canvas Click Event

In this part of the process, the cursor coordinates will be used in "getImageData" which will return a unique color. This color can then be used as a key in the dictionary object to get the correct column index.

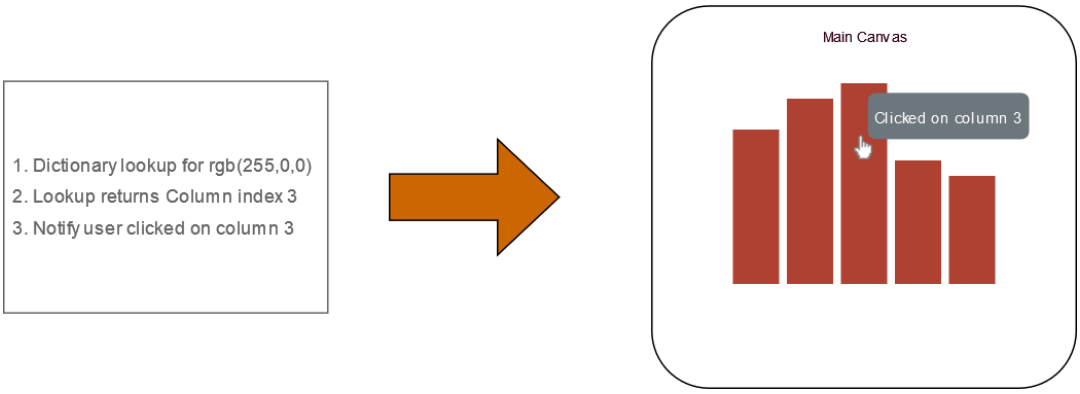


Figure 10 Dictionary Lookup on Canvas Colors

The dictionary lookup returns the index of the columns which can then be passed onto the user. The popup can be through notification, popup, or other method.

To reiterate how the whole process functions, the main steps will be listed below:

1. Render two canvases on a webpage/html document, one is visible (main canvas), and one is invisible (hidden)
2. Generate or gather the data for the canvases
3. For each entry of data generate a unique color and populate it onto the hidden canvas
4. Store the unique colors and their index onto a dictionary object
5. Attach an event listener onto the main canvas, for example "onclick" or "onhover" events
6. Grab the user's cursor event coordinates on the canvas

7. Use “getImageData” with cursor parameters to retrieve the hidden canvas’ column color. “getImageData” works in this scenario since both canvases are of same size
8. Refer to the dictionary object to get the column information using the unique color as a key

3.2 Data Fetching Upgrade

The M-Sim tool had previously been using ElasticSearch scroll function, which is not the recommended method for deep pagination. The documentation on ElasticSearch has stated to use search-after instead [57]. In this chapter, I will discuss pagination in ElasticSearch explaining “from + size” pagination, scroll query, and search-after query.
The appropriate pagination request needed for users can be determined by user behavior. Are users traversing through a large array of elements, perhaps a table of inventory in an online store, or instead, a small list? Depending on the size of elements – the most appropriate pagination request can change. Performance of both requests play an important part on when to use which specific request. In the ElasticSearch environment, a smaller table or list would be one with less than or at most 10.000 elements. For this size, appropriate pagination requests include the “from-to-size” pagination request. This pagination request works as mentioned in introduction section, by slicing the list of elements from the desired point. The “from” keyword points to which element should the return list start at. The “size” keyword will specify how many elements to return.
To work with ES there are several available tools, one of which is called Kibana. Kibana is a visualization tool, which can create requests or queries to the ES API, which display corresponding responses. These same requests can be executed in M-Sim tool and chrome developer tools can be used to analyze information on the requests.
Before we discuss pagination methods, it is important to understand scroll context in ES. Scroll uses context, which maintains a state of all the documents which matches requirements stated by a given query. The list of documents can contain documents, which may be currently opened or in use by other users. To prevent modification from other users, scroll queries contain a keep alive parameter which tells ES how long the scroll context should be active for. There are also background processes which optimizes index by merging smaller segments to create new and bigger segments. But open scroll context prevents the older and smaller segments from being deleted. Resulting in more disk space and file handles being required. There is also a limit to how many open scroll contexts there can be, which is 500. It is also recommended to allocate sufficient heap space in nodes according to the documentation [16].

3.2.1 Pagination

3.2.1.1 From + Size

In ES the “from + size” or cursor-based pagination is well suited for paginating document tables with at most 10.000 elements. Beyond that it is referred to as deep pagination which cannot be handled by “from + size”. The problem with using “from + size” pagination starts early since search requests span through multiple shards [16]. Moreover, each of these shards load up its own query results and results from previous pages in memory. For large sets of results these factors can combine to degrade performance as well as cause node failures due to the depletion of sources.
Below is an image of a “from + size” search request (query parameters are censored for confidential reasons).

Figure 11 From + Size Query

As mentioned above, performance is degraded and this happens when there are not enough resources, and due to poorly structured queries. Search requests with “from + size” take heap space and time proportional to “from + size” queries and this will limit memory [56]. In other words, suppose there are two queries, query A and query B. Query A has parameters of “from 0, size: 100” while query B is structured as “from: 5000, size: 5000”. Typically, Query A should perform better than query B with parameters “from: 5000, size: 5000”. However, that is not always the case. As can be seen in appendix where memory consumption of query B consumes less memory (A1). Although the timing of these requests reflects that the Query A is more efficient than Query B due to the lower total time to fulfill request.

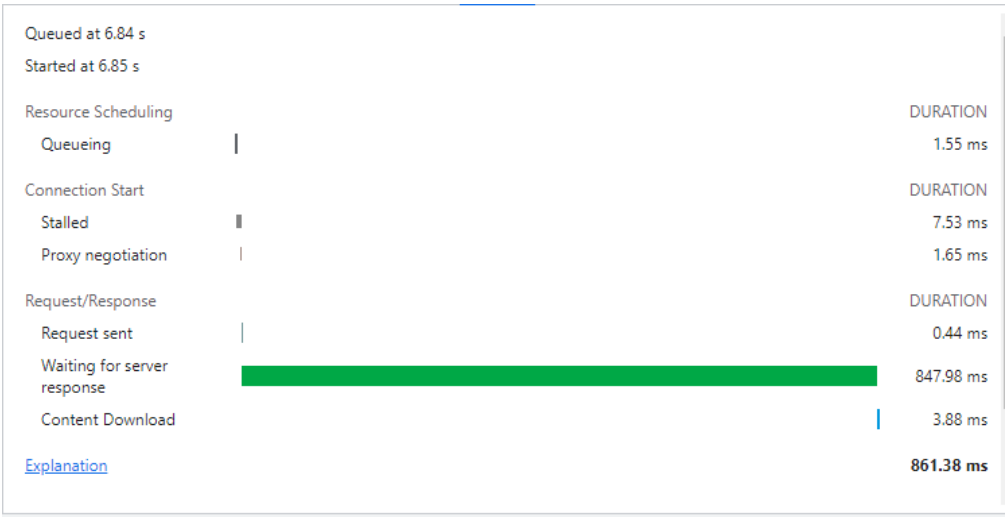


Figure 12 From 0th Element, Size 100

Figure 13 From 5000th Element, Size 5000

3.2.1.2 Scroll Request

Moving on, if we wish to paginate a larger list, suppose 100.000 elements the type of requests would include scroll and search-after. Scroll request and search-after pagination is different than typical “from-to-size”. Since these requests will need a unique value as a frame of reference when creating pagination requests in order to get the batches of data that are next in line. To create scroll requests, an initial request is made to the ES server to get a scroll id. Response from the server will contain a scroll id as well as results from the initial query. The transactions between user and ES server can be seen from the image below.

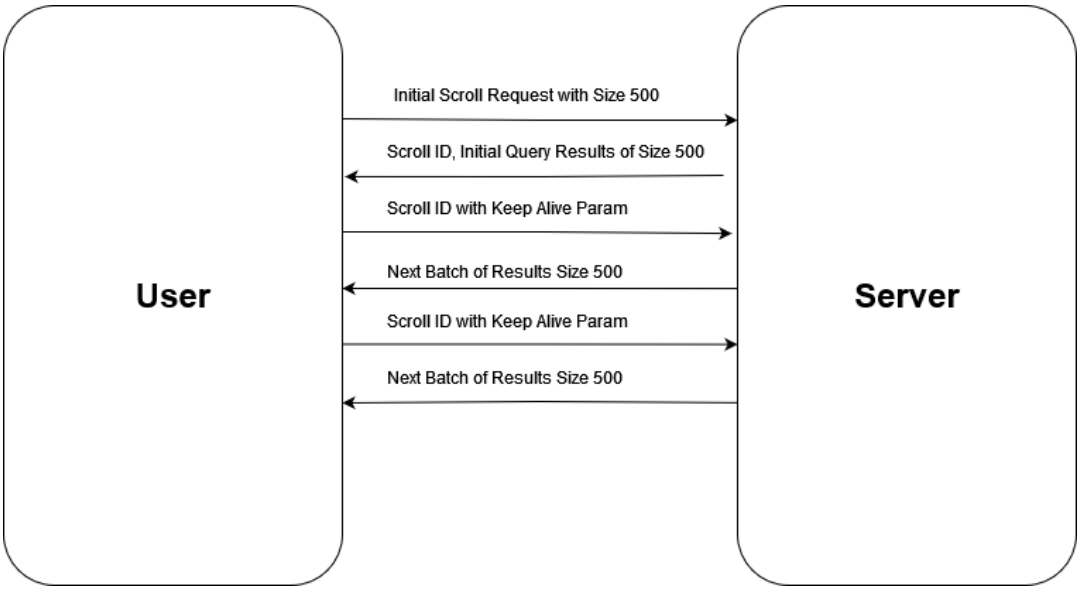


Figure 14 Scroll Request Interaction

The image above depicts how scroll queries/requests are used with ES. First an initial request to get a scroll ID, which acts as a key to unlocking the rest of the desired information, so long as the scroll ID is kept alive. The process goes on until the all the data matching the query is fetched, or until the user keeps the scroll query alive. But this comes at a cost since as long as scroll contexts remains open, more memory is consumed. Leading to the performance of the system is slowed down.

Figure 15 Initial Scroll Request

The image above shows the timing. The timing of the initial scroll request is roughly the same as a “from + size” request. But there is the added of benefit to add subsequent requests, such as the “Next batch of results” from Figure 16. These requests are shown below.

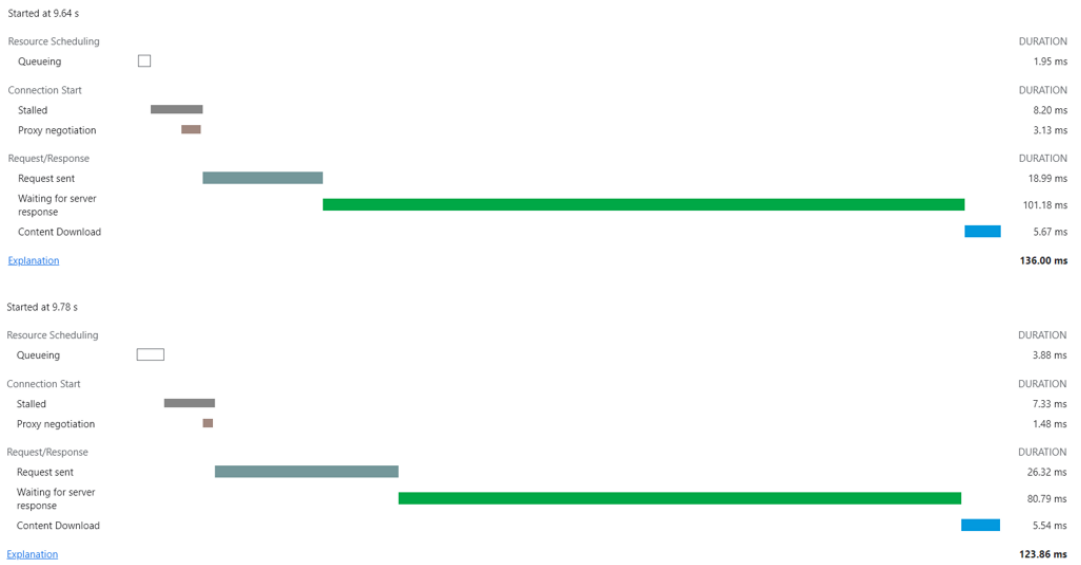


Figure 16 Subsequent Scroll Requests

The subsequent requests are relatively quick in comparison to initial scroll request. Additionally, from appendix (A1) it can be noticed that with the toy example, the size of data in memory is not too large. The problem occurs when the size of the data being read is in the range from hundreds of thousands to millions.

3.3 Search After

Search-After (SA) is another pagination technique, which shares similarities to scroll query. SA queries are stateless meaning that when querying data, it is possible that updates to the target data are made [16]. So, when one user, user 1, is in the process of fetching data, other user(s) may modify the data. Leading to user 1 fetching data that is modified. This is appropriate in real-time data analytics, but if one wishes to maintain the state of data such that there are no changes once a SA request is made, it is possible to add a Point-in-Time attribute to the search query. Point-in-time (PIT) is a view into the target data as it was when the search query was initiated. PIT ID values needs to be created before a search query can use it to maintain state of target data [59]. Similarly, to scroll query, PIT prevents background process of merging in ES. A shard in ES is a Lucene index, and these indexes are further divided into segments. Segments are internal storage elements which stores the index that contains document data [61]. In typical queries, smaller segments are merged into larger ones while the small segments are deleted to free up resources, or to optimize the system. Both scroll context and PIT creation prevents this merging process which creates overhead and more resources to be consumed. However, PIT is a more lightweight approach to maintain the state of data when compared to scroll context [16].

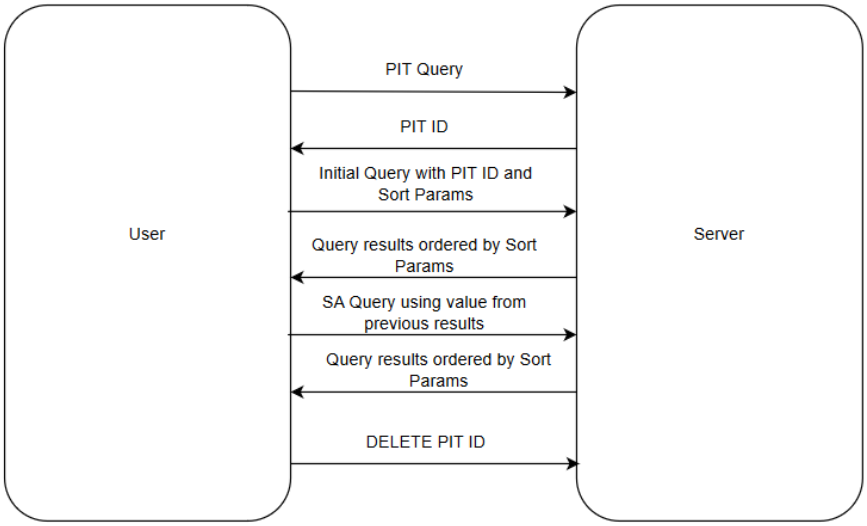


Figure 17 Search After Request Interaction

The above image shows the process of SA query, which shares multiple characteristics to scroll query. One of the differences being that the initial query does not contain any parameters rather a simple post request to an index with a PIT field specified. Since, there is no initial query, the response just contains a PIT ID. The PIT ID is then used on the first search query and no index is required to be specified as all the context for the search is stored in the PIT ID. The responses from the first search query (query number 3 on the figure above) will include sorting parameters specified by the search query. These sorting parameters need to be unique values, usually a timestamp value or in the internal shard doc value, which is included implicitly. The data matching the search query will include the matching results/hits, which will be referred to as initial set, includes sorting parameters. To use SA to get the next batch of results, simply use the last sorting parameter values in the initial set. These values act as a frame of reference to get the subsequent set of data and are included in the SA query field called “search after”. (See appendix Figure A9 for sorting values, and A10 for search-after query using sorting values from A9).

3.4 Data Fetching Problem in M-Sim

Even though the performance of scroll requests, based on Figures 16, does not seem bad. The load on the server as well as resources consumed by the

query affects the M-Sim application negatively. On contrary, SA queries do not stress the server as much and consume fewer resources.

Figure 18 Initial Search After Request

Even though performance of scroll requests can cause performance deterioration, according to testing with small queries, there were no significant changes in performance between SA and scroll queries. According to appendix A6 and A7, the performance difference between scroll and SA queries can be seen. Further testing has shown similar results, where SA queries consume more memory on the browser side, but on average take fewer requests than scroll queries. In appendix, a small test run containing 23.000 results took 10 scroll requests to load all the data. But, for SA query only 9 requests were created. Testing with bigger test runs, the difference in number of requests using scroll method when compared to SA showed an increase in size based on the size of the test runs. Resulting in large test runs to create more requests when using scroll rather than SA queries.
Additionally, in M-Sim tool using scroll query places a burden in the backend side of the environment. Therefore, an artificial limit is set on scroll query as well. This also provides conditions to update the data fetching protocol to use SA queries as an alternative. According to chrome developer tools, SA query on average consumes more memory than scroll, however, manages to fetch desired data without placing stress in the system.

3.4.1 Implementing Search After

In order to use search after in M-Sim tool there were several obstacles that had to be completed beforehand. These included using Angular framework and familiarizing with the M-Sim environment. Angular also uses RXJS, which stands for Reactive JavaScript, which include a special feature called observables. Observables can be used to manage and structure ES queries. From Figure 17 it can be seen there is an initial query to create a PIT id and then subsequent queries can be made using the PIT id. To replicate the query transactions in Figure 17, there would be two observables made, one for the initial query and another for the subsequent queries. RXJS have special functions called “subscribe”, “expand”, and more. These functions can be used in

conjunction to achieve complex functionality, which can produce SA queries. By subscribing to the data-stream of the initial PIT query, ES server sends to web client the initial results/hits. It is possible then to use “expand”, which is similar to subscribe, but uses an index to iterate itself so you can subscribe to the query results more than once.
As mentioned earlier search after queries consume less resources and create fewer requests (see appendix A5 and A6), although this can depend heavily on how the code for fetching data is configured. Within M-Sim tool, however, SA queries were the preferred method to retrieve data rather than scroll queries. With the current architecture or implementation of M-Sim tool, queries are needed to execute so log or transaction data can be fetched and displayed. These queries are executed in parallel fashion, the number of these queries as well as how many queries will be executed over time have not been taken into account. However, M-Sim will continue to grow resulting in more and more queries needing to be executed. But due structure of M-Sim tool, there was no internal handling on queries and their search context limit. Eventually, as the number of open search contexts (created by scroll queries) exceed the limit, data fetching queries will begin to fail. While scroll queries could be managed with M-Sim tool if there was handling for open search contexts, this is extra overhead. On the other hand, SA queries do not have a limit for their search contexts, which leads M-Sim tool to begin using SA queries as their method for data fetching.

4 Conclusions

The results from this paper have explored and answered both of my questions when it comes to visualizing data on a webpage environment. The considerations with regards to displaying large amount of data include memory consumption, ease-of-use, responsiveness, and overall user experience. Certain expectations should be reached before deciding the data visualization method.

The main expectation was that it is relatively easy to implement for those are familiar with front-end web development, or JavaScript. These were the considerations when it comes displaying a large amount of data. Finally, which methods exist at present to visualize data based on the expectations – firstly, WebGL was eliminated as a choice since it is not simple to implement and use. Several other frameworks, which are simple to implement, are D3.js and Apache Echarts, where the latter has already been chosen to be implemented within M-Sim tool. Apache Echarts uses both canvas and SVG for visualizations. Apache Echarts uses canvas for visualizing datapoints in the magnitude of millions. However, the custom approach for visualizing large quantity of data (chapter 3.1), the method detailed in Yannick Assogba's blog was used. The technique uses two canvas elements where one is hidden while the other is visible. The visible (main) canvas would be the data visualization while the hidden canvas is coloured with unique colours for each pixel. When the cursor is on the visible canvas, the coordinates are stored and mapped onto to hidden canvas. The hidden canvas element uses the mouse coordinates as a key value to find the pair value, the column value, or data value in the data visualization. This method creates a seamless experience for interacting with data.

The final obstacle was upgrading the fetching protocol in M-Sim for receiving more data. The current implementation (scroll query) was outdated since it used more resources. Through Angular and RXJS it was possible to replace the scroll query relatively quick with no compromises. The final implementation of SA query using RXJS operators solved the limit issue for data received as well as reduce the consumption of resources and strain placed on the M-Sim tool.

5 References

1. Z. Hua Liu, et al (2014), “JSON data management: supporting schema-less development in RDBMS,” Association for Computing Machinery, New York, NY, USA, 1247–1258, doi: https://doi.org/10.1145/2588555.2595628
2. “What is a Schemaless Database?,” MongoDB, https://www.mongodb.com/unstructured-data/schemaless (accessed Jun. 20, 2022)
3. N. Karevoll, “An Introduction to Elasticsearch Mapping | Elastic Blog,” Elasticsearch, from: https://www.elastic.co/blog/found-elasticsearch-mapping-introduction accessed (Jun. 20, 2022)
4. V. Schreibmann, and P. Braun, (2015). Model-driven Development of RESTful APIs. In Proceedings of the 11th International Conference on Web Information Systems and Technologies - WEBIST, ISBN 978-989-758-106-9; ISSN 2184-3252, pages 5-14. DOI: 10.5220/0005411200050014
5. A. Haas, J. Kummerow, and A. Zakai, “Up to 4GB of memory in WebAssembly,” V8.Dev, https://v8.dev/blog/4gb-wasm-memory, (accessed July 01, 2022).
6. “SVG 1.1”, W3C, https://www.w3.org/TR/SVG11/intro.html, (accessed July 02, 2022).
7. Upper limit on number of rows?, Github Inc., https://github.com/bvaughn/react-window/issues/108
8. K. Chang, Intro to Web Graphics with HTML5 Canvas, 2018, In: Malmö, Sweden, Proceedings of Software Developer Conference, from: https://www.youtube.com/watch?v=qVQzSJ190Ow (accessed July 01, 2022).
9. “JSON in Oracle Database”, Oracle, from: https://docs.oracle.com/en/database/oracle/oracle-database/18/adjsn/json-in-oracle-database.html#GUID-D7BCE045-EF6D-47E9-9BB2-30C01933248E (accessed July 05, 2022).
10. D. Petkovic, (2017). "JSON Integration in Relational Database Systems," International Journal of Computer Applications. 168. 14-19, doi: 10.5120/ijca2017914389 (accessed on Aug. 3, 2022)
11. “Introduction to Oracle XML DB”, Oracle, from: https://docs.oracle.com/en/database/oracle/oracle-database/21/adxdb/intro-to-XML-DB.html#GUID-AE5CB350-6B46-4805-A1A9-A73A641721D1 (accessed on Aug. 5, 2022)
12. Elmasri, R. (2018). Database Schema. In: Liu, L., Özsu, M.T. (eds) Encyclopedia of Database Systems. Springer, New York, NY, doi: https://doi.org/10.1007/978-1-4614-8265-9_80735 (accessed Aug. 5, 2022)

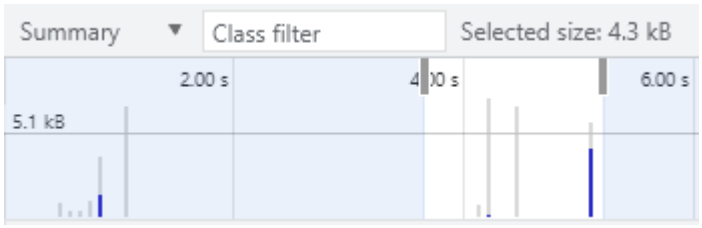
13. “What is Elasticsearch?”, Elasticsearch, from: https://www.elastic.co/what-is/elasticsearch (accessed on Aug. 9, 2022)
14. “Scroll API”, Elasticsearch, from: https://www.elastic.co/guide/en/elasticsearch/reference/current/scroll-api.html (accessed on Aug. 9, 2022)
15. E. Resnick, ““Cursor Pagination” Profile”, JSONAPI ORG, from: https://jsonapi.org/profiles/ethanresnick/cursor-pagination/ (accessed on Aug. 10, 2022)
16. “Paginate search results”, Elasticsearch, from: https://www.elastic.co/guide/en/elasticsearch/reference/current/paginate-search-results.html#scroll-search-context (accessed on Aug. 10, 2022)
17. A. Bellamy-Royds, K. Cagle, and D. Storey (2017), “Using SVG with CSS3 and HTML5,” O'Reilly Media, (accessed on Aug. 11, 2022)
18. C. Peng (2000), "Scalable Vector Graphics (SVG)," Research Seminar on Interactive Digital Media, doi: 10.1007/978-0-387-35973-1_1159 (accessed on Aug. 11, 2022)
19. R. Agrawal, A. Kadadi, X. Dai, and F. Andres (2015), “Challenges and opportunities with big data visualization”, In Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems (MEDES '15), Association for Computing Machinery, New York, NY, USA, 169–173, Doi: https://doi.org/10.1145/2857218.2857256
20. D. Fisher, (2016), “Big data exploration requires collaboration between visualization and data infrastructures”, In Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA '16). Association for Computing Machinery, New York, NY, USA, Article 16, 1–5. doi: https://doi.org/10.1145/2939502.2939518
21. S. M. Ali, N. Gupta, G. K. Nayak and R. K. Lenka, (2016), "Big data visualization: Tools and challenges," 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Greater Noida, India, 2016, pp. 656-660, doi: 10.1109/IC3I.2016.7918044.
22. J. Heer and S. Kandel (2012), Interactive analysis of big data, XRDS 19, 1 (Fall 2012), 50–54, doi: https://doi.org/10.1145/2331042.2331058
23. Lazy Initialization, Microsoft, https://docs.microsoft.com/en-us/dotnet/framework/performance/lazy-initialization (accessed on Oct. 20, 2022)
24. Loring, M. et al., Semantics of Asynchronous JavaScript, Proceedings of the 13th ACM SIGPLAN International Symposium on on Dynamic Languages, 2017, pp. 51-62, doi: https://doi.org/10.1145/3133841.3133846
25. Okamoto, S. and Kohana, M., Load distribution by using Web Workers for a real-time web application, International Journal of Web

Information Systems, 2011, Vol. 7 No. 4, pp. 381-395, doi: https://doi.org/10.1108/17440081111187565
26. NS C., Benchmarking React Library: A Developer Perspective, Dublin Business School, Dublin, 2021, Available at: https://esource.dbs.ie/bitstream/handle/10788/4277/msc_siddalingaswamy_cn_2021.pdf?sequence=1&isAllowed=y
27. Web Workers API, MDN, Available at: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API
28. Virtual DOM and Internals, Facebook Open Source, Available at: https://reactjs.org/docs/faq-internals.html
29. NgZone, Google 2012-2022, Available at: https://angular.io/guide/zone
30. Data-Driven Documents, Mike Bostock, Available at: https://d3js.org/
31. Nair, L., Shetty, S. & Shetty, S. (2016). Interactive visual analytics on Big Data: Tableau vs D3.js. Journal of e-Learning and Knowledge Society, 12(4),. Italian e-Learning Association. Available at https://www.learntechlib.org/p/173675/
32. Avoid an excessive DOM size, Chrome Developers, Available at: https://developer.chrome.com/docs/lighthouse/performance/dom-size/
33. Scalable Vector Graphics (SVG) 1.0 Specification, 2001, W3C, Available at: https://www.w3.org/TR/2001/REC-SVG-20010904/REC-SVG-20010904.pdf
34. All About Images, University of Michigan Library, Available at: https://guides.lib.umich.edu/c.php?g=282942&p=1885352
35. Su C.Y and C., Chang R., Liu J.C, Recognizing Text Elements for SVG Comic Compression and Its Novel Applications, Proceedings of the International Conference on Document Analysis and Recognition, 2011, ICDAR. 1329 - 1333. 10.1109/ICDAR.2011.267., Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.888.3190&rep=rep1&type=pdf
36. Fulton J., Fulton S., HTML5 Canvas: Native Interactivity and Animation for the Web 2nd ed, California, USA, O'Reilly Media Inc., 2013, ISBN-10 9781449334987
37. Garaizar P., Vadillo M., López D., Benefits and Pitfalls of Using HTML5 APIs for Online Experiments and Simulations, International Journal of Online Engineering (iJOE) 8, 20-25, 2013, 10.3991/ijoe.v8iS3.2254.
38. Kee D., Salowitz L., Chang R., Comparing Interactive Web-Based Visualization Rendering Techniques, Tufts University, Available at: https://www.cs.tufts.edu/~remco/publications/2012/InfoVis2012-Poster-RenderingTechniques.pdf

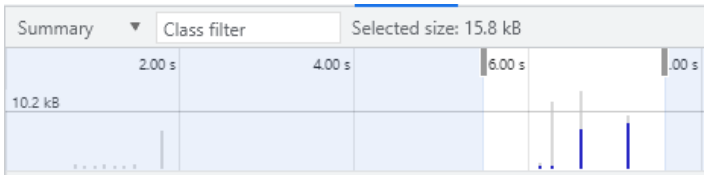
39. Eriksson Kuitu J., Visualizing public transport with heat-maps: Comparing the scalability of SVG and Canvas for heat-maps, Bachelor's Thesis Mid Sweden University Department of Computer and System Science, 2020, Available at: https://miun.diva-portal.org/smash/record.jsf?pid=diva2%3A1397765&dswid=2857
40. Sauerwein T., Evaluation of HTML5 for its Use in the Web Mapping Client OpenLayers, Bachelor's Thesis Zweibrücken University of Applied Sciences, 2010, Available at: https://www.scribd.com/document/96680799/Tobias-Sauerwein-Evaluation-of-HTML5-for-Its-Use-in-the-Web-Mapping-Client-OpenLayers
41. Vaughan-Nichols S., Will HTML 5 Restandardize the Web?, Computer, vol. 43, 2010, pp. 13-15, DOI: 10.1109/MC.2010.119.
42. WebGL, Khronos Group, Available at: https://www.khronos.org/webgl/
43. Parisi T., Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages 1st Ed, California, USA, O'Reilly Media Inc., 2014, ISBN-10 1449362966
44. Apache Echarts, Apache Software Foundation, Available at: https://echarts.apache.org/en/index.html
45. Austin B., The Growth Of The Internet: From 1990 to 2019, absolute., 2019, Available at: https://absolute.digital/magazine/seo/the-growth-of-the-internet-from-1990-to-2019/
46. Cunningham, P., Cord M., Dimension Reduction. (eds) Machine Learning Techniques for Multimedia. Cognitive Technologies. Springer, Berlin, Heidelberg. ISBN 978-3-540-75171-7, doi: https://doi.org/10.1007/978-3-540-75171-7_4 25 oct
47. .mousemove(), OpenJS Foundation, Available at: https://api.jquery.com/mousemove/ 14 Nov
48. event.pageY, OpenJS Foundation, Available at: https://api.jquery.com/event.pageY/ 14 Nov
49. Element: mousewheel event, MDN, Available at: https://developer.mozilla.org/en-US/docs/Web/API/Element/mousewheel_event 14 Nov
50. Assogba Y., Needles, Haystacks, and the Canvas API, Bocoup, Available at: https://bocoup.com/blog/2d-picking-in-canvas 21 Nov
51. Basic usage of canvas, MDN, Available at: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_usage Nov 22
52. Drawing shapes with canvas, MDN, Available at: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes Nov 22
53. Prabhu T., Plotting a Bar Chart Using JavaScript and HTML Canvas, Medium, 2021, Available at: https://blog.devgenius.io/plotting-a-bar-chart-using-javascript-and-html-canvas-db082eac28aa

54. CanvasRenderingContext2D.getImageData(), MDN, Available at: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/getImageData
55. Lubbers, P., Albers, B., Salim, F, Using the HTML5 Canvas API. In: Pro HTML5 Programming. Apress, 2010, doi: https://doi.org/10.1007/978-1-4302-2791-5_2
56. Index modules, Elasticsearch B.V., https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules.html#index-max-result-window
57. Scroll API, Elasticsearch B.V., https://www.elastic.co/guide/en/elasticsearch/reference/current/scroll-api.html
58. Horrible indexing performance as index size grows, Elasticsearch B.V, https://discuss.elastic.co/t/horrible-indexing-performance-as-index-size-grows/61674
59. Point in time API, Elasticsearch B.V., https://www.elastic.co/guide/en/elasticsearch/reference/current/point-in-time-api.html
60. Refresh API, Elasticsearch B.V., https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-refresh.html
61. Merge, Elasticsearch B.V., https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-merge.html
62. Subscription, RxJS, https://rxjs.dev/guide/subscription
63. Expand, RxJS, https://rxjs.dev/api/operators/expand

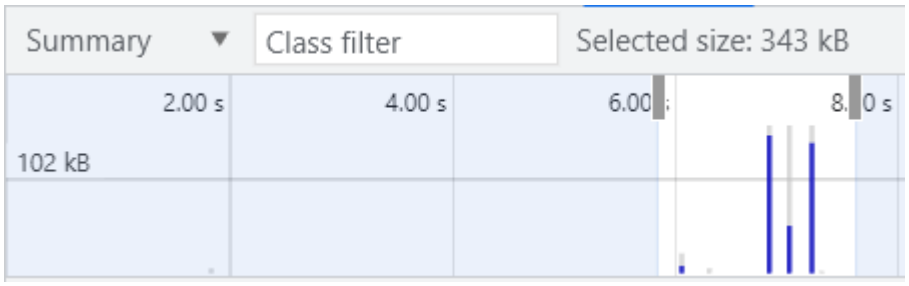
Appendix

A1: 

From Size Memory 5000, 5000

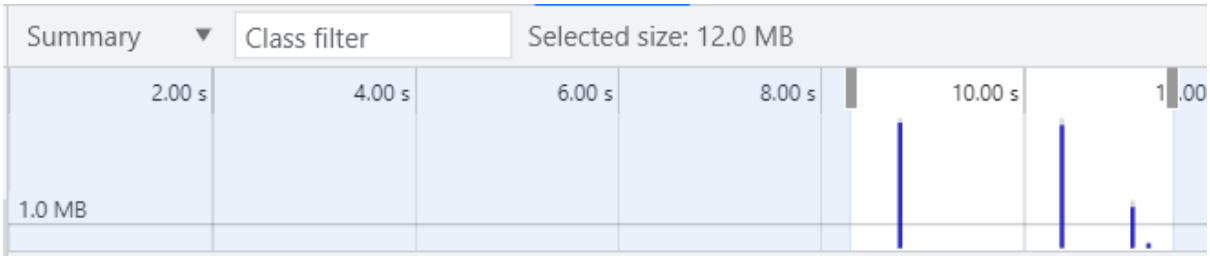
A2: 

From Size Memory 0, 100

A3: 

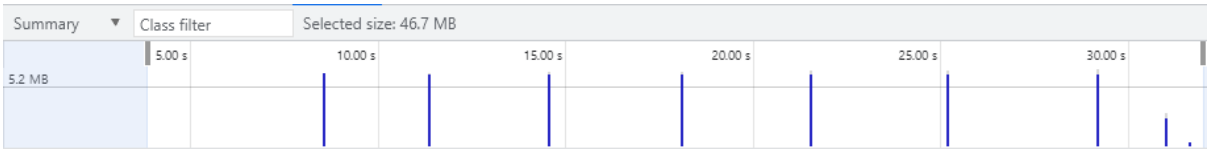
Scroll Queries Size 100, 3 iterations

A4:



Scroll All Data Memory, 23.000 hit example

A5:



Scroll All Data Memory, 73.000 hit example took 10 requests

A6:

Search After Memory, 73.000 hit example took 9 requests

A7:

```
POST /[redacted]/_pit?keep_alive=5m
```

Initial PIT request

A8:

```
POST /_search
{
  "size": 10000,
  "track_total_hits": true,
  "query": {
    "match": {
      [redacted]
    }
  },
  "pit": {
    "id" : "idValue",
    "keep_alive": "1m"
  },
  "sort": [
    {
      "timestamp": {
        "order": "asc",
        "format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis"
      }
    },
      {"_shard_doc": "asc"}
  ],
  "_source":["timestamp","metrics.*"]
}
```

Search Query with PIT

A9:

Results from Search Query with PIT

A10:

```
POST /_search
{
  "size": 10000,
  "track_total_hits": true,
  "query": {
    "match": {
      "sw_build" : [redacted]
    }
  },
  "pit": {
    "id" : "idValue",
    "keep_alive": "1m"
  },
  "sort": [
    {
      "timestamp": {
        "order": [redacted]
        "format":
      }
    },
      {"_shard_doc": "asc"}
  ],
  "_source":["timestamp","metrics.*"],
  "search_after" :  [
    [redacted]
    ]
}
```