Approach to Selecting an Appropriate Javascript Charting Library for Graphically Rich Single Page Javascript Applications

ALEN RAJŠP, GREGOR JOŠT, VIKTOR TANESKI, SAŠA KUHAR, LUKA PAVLIČ, University of Maribor

React is one of the three most popular libraries for the development of single page applications. It also enables the implementation of reusable user interface interfaces. Additional challenges and limitations in development are the use of external libraries and frameworks. The paper presents a selection method for selecting an appropriate JavaScript charting library. The steps (identification, licensing elimination, functionalities' elimination, complexity elimination and prototyping) of the method are presented briefly, and demonstrated on a use of the method for the needs of a React Single Page Application that was focused heavily on graphically intense visualizations. The selected library (C3.js) is presented, and some cases of use are shown, with accompanying screenshots that demonstrate the complexity of implemented visualizations.

1. INTRODUCTION

The web has become a de-facto platform of user-friendly user interfaces. Web applications are no longer filled with inconvenient flashing webpages that, whenever a user clicks on a button, the whole website needs to be served to the user client. The concepts that were once meant for websites are no longer viable for user-friendly web applications. Instead, web applications became exactly what they are called - applications. They operate within the web browser environment and communicate independently with the backend system. Technological solutions, such as JavaScript, REST interfaces and JSON format, are the basic components of such applications, but, for a fast and relatively sustainable development of web applications, this set is inadequate. Without the use of specialized libraries (e.g. React.js) or frameworks (e.g. Angular), we can no longer deal with the development of single page applications. Libraries and frameworks provide developers with the means to develop single page applications. At the same time, they also bring additional challenges, such as the coexistence of "classical" HTML or JavaScript content and components in a wider application, built on third-party libraries and frameworks.

We had to develop a single page application for an external client. One of the main functionalities of the application was the appropriate visualization of data that the application receives from the backend system. Data visualization was achieved with the help of a third party library, which had to be adapted to the host - the React library. The article presents our selection process of JavaScript charting libraries, and experience with the selected JavaScript charting library C3.js.

Author's address: A. Rajšp, UM FERI, Koroška cesta 46, 2000 Maribor, Slovenia, email: alen.rajsp@um.si; G. Jošt, UM FERI, email: gregor.jost@um.si; V. Taneski, UM FERI, email: viktor.taneski@um.si; S. Kuhar, UM FERI, email: sasa.kuhar@um.si, L. Pavlič, UM FERI, email: luka.pavlic@um.si

In: Z. Budimac and B. Koteska (eds.): Proceedings of the SQAMIA 2019: 8th Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, Ohrid, North Macedonia, 22–25. September 2019. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

1.1 React.js library

React.js is a JavaScript library for user interface development, developed by Facebook. Library was first used in 2011 on the social network Facebook for the news feed functionality [Education Ecosystem 2018], after which it was also used on the social network Instagram in 2012. The library became open source in 2013. The library soon became very popular for developing user interfaces among web developers [Williams 2019]. A survey on Stack Overflow [Stackoverflow 2019] for 2019, shows that the React library is at the top of the most wanted web frameworks, and the second most used web framework (after jQuery). React library is used for the View in traditional Model-View-Controller (MVC) architecture. It is based on components where each component represents a standalone module that renders a predefined structure. Modules can be reused and nested.

Unlike other JavaScript libraries or frameworks, React does not work directly with the Document Object Model (DOM). DOM is an interface that allows scripts to access and update the content, structure, and style of a document dynamically. Because DOM manipulation is ineffective, React uses virtual DOM (VDOM) [Facebook Inc.], where a virtual presentation of a user interface is saved in the memory. When components are updated, so is VDOM, and React synchronizes VDOM continuously with the DOM structure. It is important to note that only parts where changes occurred are re-rendered, and the rest stay unchanged. This allows React only minimum DOM manipulations, which allows VDOM manipulations to remain fast and efficient [Abe; Hamedani 2018].

The main feature of the React library is the non-segregation of the application's logic from its presentation to various files. For this purpose, the JSX extension is used, which represents a syntactic extension of the programming language JavaScript. It is designed to define elements.

Reacts is not a framework, but a library. Therefore, for complex applications, additional libraries are needed, e.g. for routing between individual pages, obtaining data, managing the situation throughout the application, visualizing data, etc. Selection of complementary libraries to use with React presents a special challenge.

2. SELECTION METHOD OF JAVASCRIPT VIZUALIZATION LIBRARIES

2.1 Pretext

The practical solution developed enables complex visualizations of linked data. The visualization of data was the core functionality of this application. Because JavaScript has been gaining in popularity, so have third party charting libraries and libraries for data visualizations. For our needs, the understanding of the operation and supported functionalities in popular visualization JavaScript libraries was crucial. It was necessary to select what would best suit the requirements of the client. One of the crucial factors for selection were: (1) The right set of graphs for display of data, (2) The amount of data shown on charts, (3) Display of graphs on different devices (responsiveness), (4) Browser support, (5) Ease of customization.

2.2 Requirements

The developed application was to be used on bigger screens (computers and tablets, not mobile phones), and had to allow visualization of larger amount of data. The user interface visual requirements were provided from the external client as an Adobe XD rendered wireframe prototype. One of the prototype images is shown in a screenshot below in Figure 1. The actual text content is omitted due to copyright.

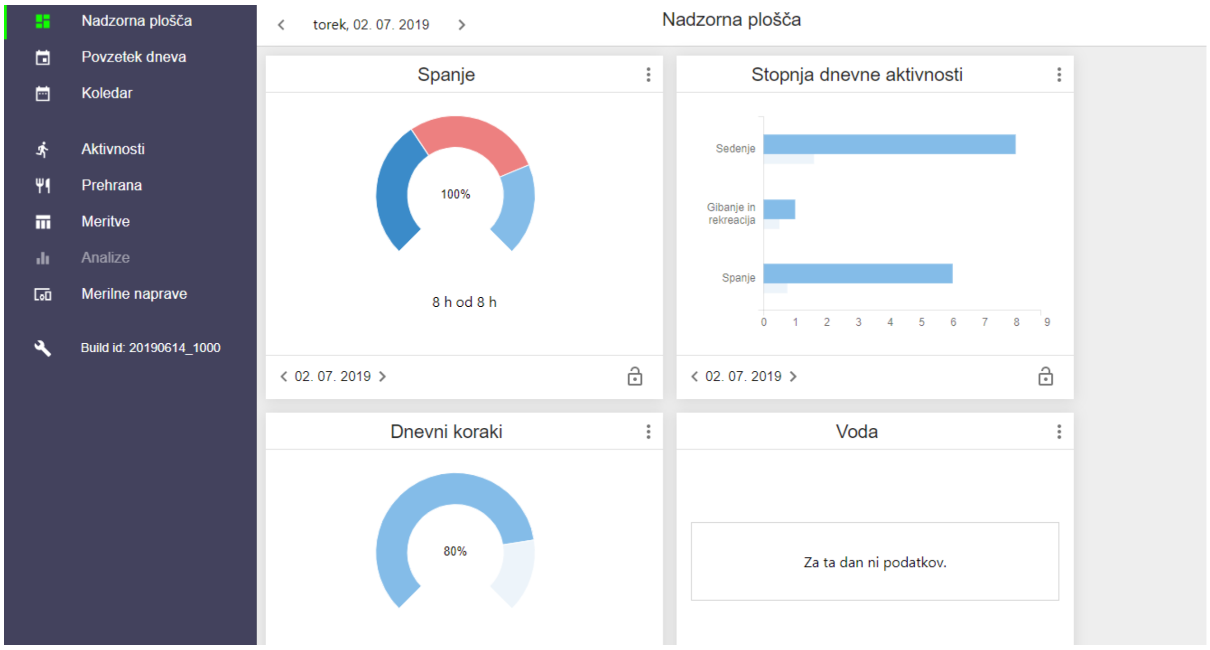


Fig. 1 Screenshot of application prototype wireframe in Adobe XD

As such, the implemented application had to be visually as similar as possible to the provided screens in the visual prototype. In addition to rather standard types of graphs (column, row, line, stacked), some heavily specialized types of graphs were also needed that had to be modified heavily e.g. overlapping row charts (Figure 2), stylized gauge charts (Figure 3), stacked column charts (Figure 4), column charts with overlay region (Figure 5), bar charts combined with stacked bar charts below (Figure 6).

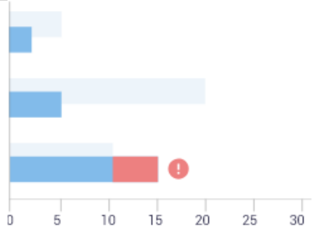


Fig. 2 Overlapping column chart with icon

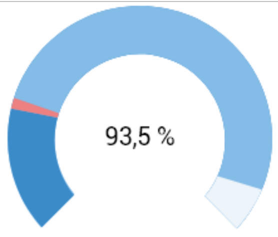


Fig. 3 Stylized gauge chart

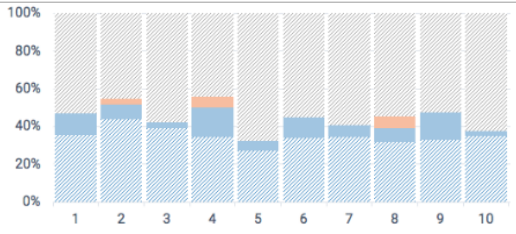


Fig. 4 Stacked column chart

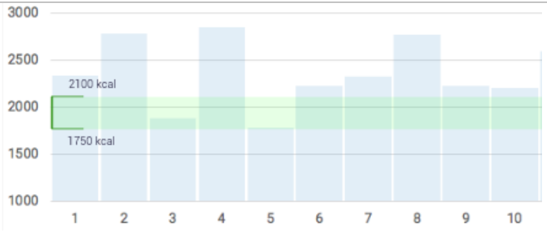


Fig. 5 Column chart with overlay region

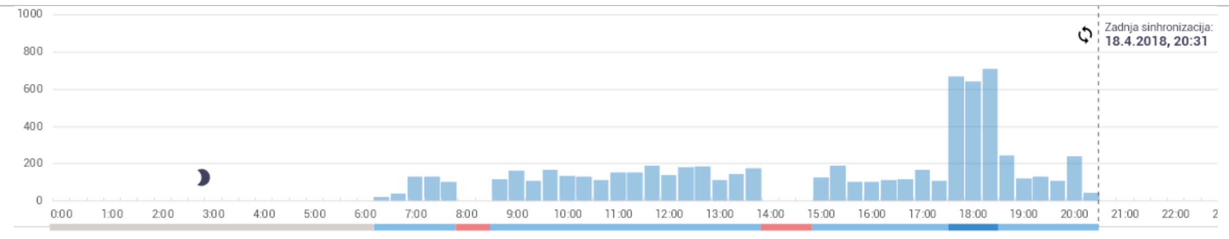


Fig. 6 Bar chart combined with stacked bar chart beneath

2.3 Selection method

Multiple formal methods for software selection already exist, as presented in [Jadhav and Sonar 2009]. Their systematic literature review proposes a generic [Jadhav and Sonar 2009] seven step methodology at selecting software. The steps roughly translate to (1) Determining the need for purchasing the software, and preliminary research of availability on the market, (2) Shortlisting of candidates, (3) Eliminating most candidates based on features, or which do not work with existing systems, (4) Evaluation techniques to evaluate remaining packages and scoring, (5) Trialing top software packages, pilot testing the candidates, (6) Negotiating a contract, (7) Purchase and implementation.

Since JavaScript software packages contain hundreds of dependencies, such rigorous process needs to be updated. We wanted to ensure that the system could be updated in the future, and maintain the possibility of introduction of new functionalities. The selection process goal was to select an appropriate visualization library. In our selection criteria, it was important that the library enabled implementation of the required graph types, was open source, free to use in commercial applications, well documented, regularly updated, as simple as possible to use, and to allow for adjustments.

As shown in Figure 7, we created the following selection process for selection of JavaScript libraries. In comparison with the previously presented formal selection methodology, we eliminated the steps based with contract negotiation, since they are not as applicable with JavaScript dependencies.

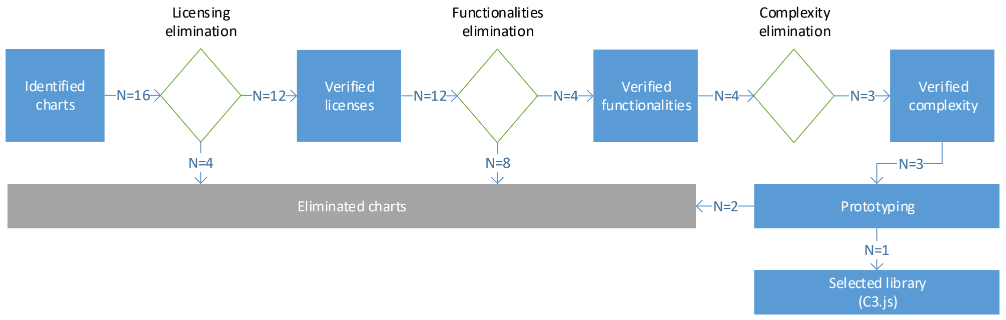


Fig. 7 JavaScript visualization library selection process

Libraries' overview

Identified charts are shown below in Table 1. The chosen library **C3.js** is marked by a trophy icon (🏆) and libraries selected for prototyping phase by a check icon (✔). Regarding the identification criteria at each step, we've taken inspiration from the publicly posted comparison available on Wikipedia [Wikipedia 2019].

Table 1. Overview of identified Javascript charts

Selected	Prototyping	Library	GitHub stars (11. 2018)	License	Implementation complexity	Customizability	Missing graph types	Missing functionalities
		Chart.JS [Chart.js]	40361	MIT [Opensource.org]	low	low	3	4
		Chartist.JS [Chartist]	11231	MIT	low	low	2	5
🏆	✓	C3.js [C3.js]	8089	MIT	low	medium	2	1
		D3.js [Bostock]	80246	BSD-3 [Open Source Initiative]	high	high	0	0
	✓	Recharts [Recharts]	10318	MIT	low	medium	2	2
		Plotly.js [Plotly.js]	9282	MIT	low	medium	2	2
		NVD3.js [NVD3]	6730	Apache 2.0 [ASF]	low	medium	3	5
		Victory [Formidable Labs]	6548	MIT	low	low	3	4
		Vx [Shoff]	5311	MIT	medium	high	3	5
		React-Vis [Uber Open Source]	4696	MIT	low	low	3	4
		Britecharts [Eventbrite]	3184	Apache 2.0	low	low	3	5
	✓	Incubator echarts [Apache]	31145	Apache 2.0	medium	medium	2	2
		AnyChart [AnyChart]	188	proprietary	Libraries that are not free for commercial use			
		Amcharts [Amcharts]	373	proprietary				
		CanvasJS [Fenopix]	9	proprietary				
		Google Charts [Google]	763	proprietary				

Identification

The first phase is identification of candidate libraries. This is best done by researching published libraries and dependencies on source code repositories (e.g. GitHub) and package manager websites (e.g. NpmJs [NPM Inc]). These sites are beneficial because they provide transparent statistics about the libraries. We can see how many times specific libraries have been downloaded, how much and when they have received updates, how many developers use them, etc. We used the code repository GitHub [Microsoft] for identification of charting libraries by querying the following phrases "Charts" and “Graphs” independently; the search was conducted in November 2018. We filtered the results by programming language to JavaScript and selected the 16 most popular (most stars on GitHub).

License elimination
In license elimination we eliminated all the libraries where licensing is inadequate for specific projects. It is important to identify the licensing of each identified library in order not to waste resources with invalid libraries. In our case, this meant eliminating libraries where the whole source code was not publicly available, or that their use wasn’t permitted or free in commercial applications. Of the 16 libraries identified 50% (8) used MIT license, 19% (3) used Apache 2.0 license, 6% (1) used BSD-3 license, and 25% (4) used proprietary licensing and were eliminated from this phase.

Functionalities’ elimination
In this phase, we identified the needed functionalities of our project and the functionalities of the selected libraries, and compared them. The source for identification of functionalities is, in most cases, the official documentation. In our case, we identified functionalities of the initial selection with the supported types of graphs. We identified 13 basic types of graphs that appeared in libraries (underlined are those that were required in our implementation): Gauge, column (normal, stacked), line (normal, stacked), radar, donut, pie, scattered, funnel, gantt, quartile, polar, and bubble. From the requirements, we also identified 5 specific functionalities that we needed: Overlapping regions, icons on graphs, combination of graphs and vertical / horizontal lines with added text. Thus, we selected 4 potentially suitable libraries (C3.js, D3.js, Recharts and Incubator echarts).

Complexity elimination
In this phase, prerequisites are analyzed (mostly on what other libraries does the library depend) for using a library and difficulty of implementation. Complexity of implementation was measured by the quality of documentation and length of code needed for simple use cases.

In complexity elimination we eliminated the D3.js library from the remaining four libraries, due to severely higher implementation complexity than in the remaining cases. This was because D3.js is a visualization library, not a charting one. This means that to create charts in D3.js one would have to implement each, even the most basic building block, by himself. This means that D3.js does not come with any of the prebuilt charts that could be modified further. C3.js, on the contrary, is built on top of D3.js, which means that basic graphs are already built, but transformations possible on D3.js are also possible in C3.js.

Prototyping
In the prototyping phase we prototyped the top libraries by implementing the same cases in all of them, to select the most suitable library from the working prototypes.

In our project, the purpose of the prototyping phase was the implementation of more complex types of graphs on static web pages. An example of an attempt to draw an overlapping column graph is shown in Figure 8. In the other cases, the C3.js library also made it possible to achieve the greatest approximation to the required appearance, depending on the requirements. That is why we selected C3.js as the most suitable for our implementation.

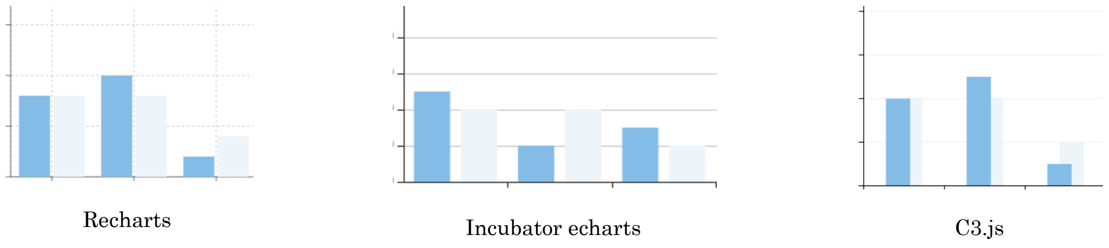


Fig. 8 Prototypes of the Recharts, Incubator echarts and C3.js charting libraries

3. VISUALIZING DATA WITH THE C3.JS LIBRARY

C3.js [C3.js] is a graphical drawing library that is licensed under MIT [Opensource.org]. This allows us to use it without any compensation in commercial projects. It is based on the library D3.js [Bostock]. The library was first published in 2014, and has already received 104 issues so far, indicating that it is topical and up-to-date. Graphs like the D3.js library are drawn in the SVG format. To use, we need to include its stylish template (CSS), the D3.js library, and the JavaScript library file.

The library divides different data types according to their use in graphs. It divides them into data, axes, legends, grids, overlay regions, each of the data types represents an object that is included in the root JSON object. Existing graphs can also be updated without creating a new instance of the graph.

Implemented graph visualizations are shown in Figures 9 and 10.

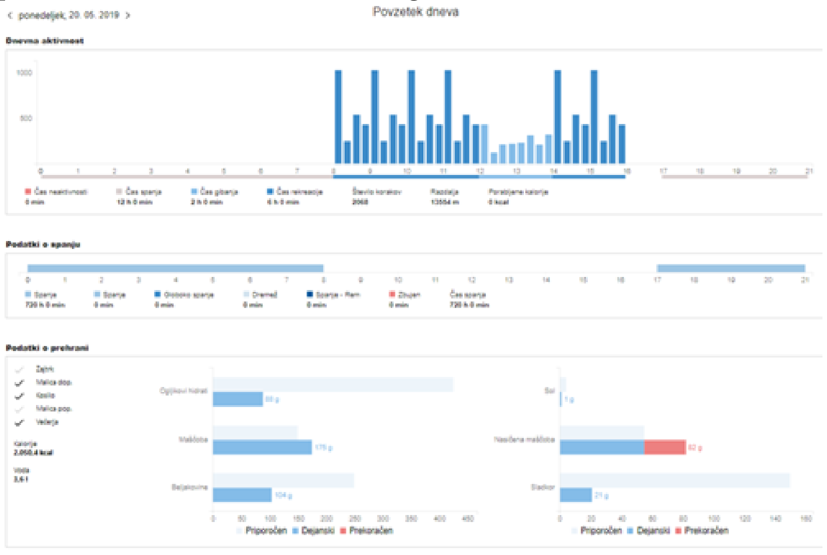


Fig. 9 Implemented visualizations on “Daily summary”

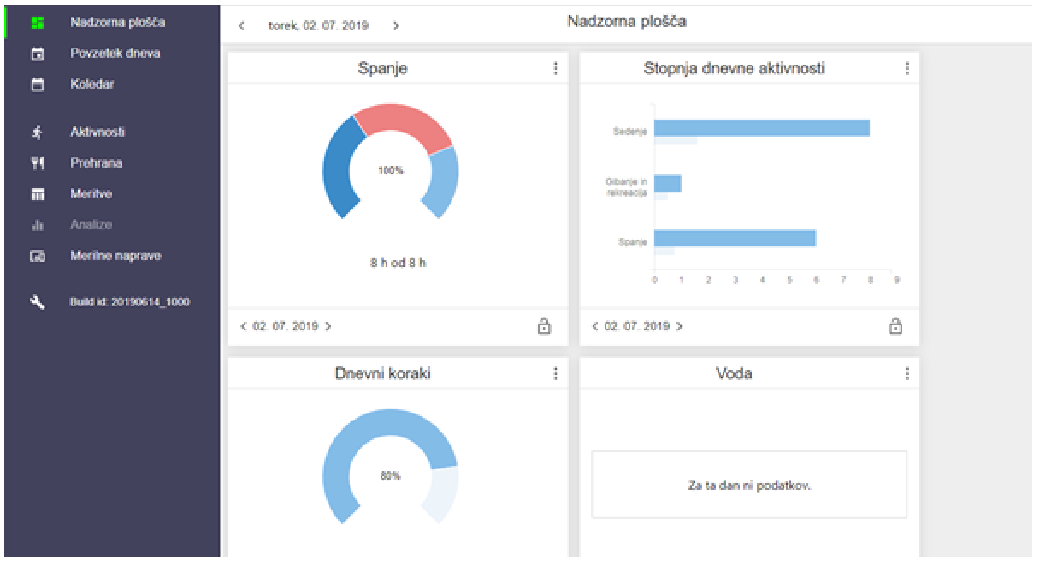
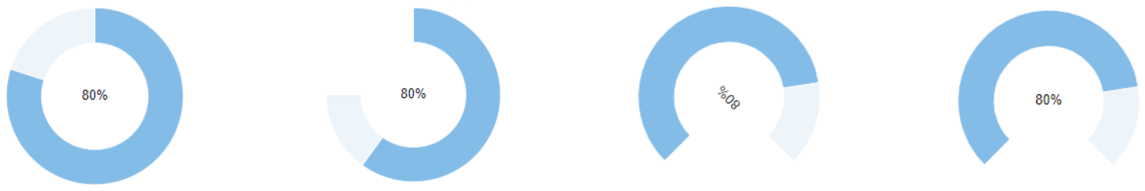


Fig. 10 Implemented visualizations on "Dashboard"

3.1 Graph adjustments

The basic graphs had to be modified so that they would meet the requirements visually and functionally. The adaptations concerned both style and functionality. Stylistic modifications were achieved by modification of stylesheets (CSS). The gauge graph (which is not a part of the prebuilt graphs in C3.js) required (Figure 11) to fill a part of the pie chart with a transparent color (2), which was done by inserting virtual data with the same color as the background, then, using the CSS transformation, the graph was rotated (3), and the words had to be rotated in the opposite direction in order to maintain normal orientation (4).



(1) Starting donut chart without modifications (2) Inserting invisible data (3) CSS graph rotation (4) CSS rotation of text in the opposite direction

Fig. 11 Transforming a donut chart in a gauge chart

In places where CSS customizations were not sufficient, we also used the methods (e.g. init - on initialization, onrendered - upon drawing the graph, onmouseover - when the cursor enters the graph of the graph, onmouseout - on leaving the cursor from the area of the graph, onresresize - while changing the size of the browser window) of the C3.js library that are provided for such specialized cases. Use of C3.js methods for transformations is much slower than use of CSS transformations, but also allows use the full use of D3.js library methods, so they were only used where CSS transformations were inadequate tools for the required results.

4. CONCLUSION

Our method for selecting the JavaScript visualization library was deemed appropriate, and its output – the selected library C3.js, was successful in implementing all the required visualizations. The selection process ensured that the selected library was compliant with all the functional and nonfunctional demands, was versatile and successful with implementing the required charts. We deem the process used in this special case to be useful also for selection of other JavaScript libraries, since it was lightweight, domain independent and simple to follow. The exact boundaries on which libraries to eliminate on each step were not mentioned, because they are specific for each project and should be selected accordingly.

REFERENCES

Abe, M. React is a JavaScript Library for Building User Interfaces; Not a Framework Like Angular. https://www.codemag.com/Article/1809041/Demystifying-React.

Amcharts. JavaScript Charts &amp; Maps - amCharts. https://www.amcharts.com/.

AnyChart. AnyChart is a lightweight and robust JavaScript charting library. https://www.anychart.com/.

Apache. ECharts. http://echarts.apache.org/.

ASF. Apache License, Version 2.0. https://www.apache.org/licenses/LICENSE-2.0.html.

Bostock, M. D3.js - Data-Driven Documents. https://d3js.org/.

C3.js. C3.js | D3-based reusable chart library. https://c3js.org/.

Chart.js. Chart.js | Open source HTML5 Charts for your website. https://www.chartjs.org/.

Chartist. Chartist - Simple responsive charts. https://gionkunz.github.io/chartist-js/.

Education Ecosystem. 2018. React.js History - Education Ecosystem. https://www.education-ecosystem.com/guides/programming/react-js/history.

Eventbrite. Britecharts - D3.js based charting library of reusable components. http://eventbrite.github.io/britecharts/.

Facebook Inc. Virtual DOM and Internals – React. https://reactjs.org/docs/faq-internals.html.

Fenopix. Beautiful HTML5 JavaScript Charts | CanvasJS. https://canvasjs.com/.

Formidable Labs. Victory | VictoryChart. https://formidable.com/open-source/victory/.

Google. Charts | Google Developers. https://developers.google.com/chart/.

Hamedani, M. 2018. React Virtual DOM Explained in Simple English - Programming with Mosh. https://programmingwithmosh.com/react/react-virtual-dom-explained/.

Jadhav, A.S. and Sonar, R.M. 2009. Evaluating and selecting software packages: A review. Information and Software Technology 51, 3, 555–563.

Microsoft. GitHub. https://github.com/.

NPM Inc. npm | policies | terms. https://www.npmjs.com/policies/terms.

NVD3. NVD3. http://nvd3.org/.

Open Source Initiative. The 3-Clause BSD License | Open Source Initiative. https://opensource.org/licenses/BSD-3-Clause.

Opensource.org. The MIT License | Open Source Initiative. https://opensource.org/licenses/MIT.

Plotly.js. plotly.js | JavaScript Graphing Library. https://plot.ly/javascript/.

Recharts. Recharts. http://recharts.org/en-US/.

Shoff, H. vx | visualization components. https://vx-demo.now.sh/.

Stackoverflow. 2019. Stack Overflow Developer Survey 2019. https://insights.stackoverflow.com/survey/2019.

Uber Open Source. react-vis. https://uber.github.io/react-vis/.

Wikipedia. 2019. Comparison of JavaScript charting libraries. https://en.wikipedia.org/wiki/Comparison_of_JavaScript_charting_libraries.

Williams, O. 2019. Making the business case for React in 2019 – LogRocket. https://blog.logrocket.com/making-the-business-case-for-react-in-2019-74463bbb22de.