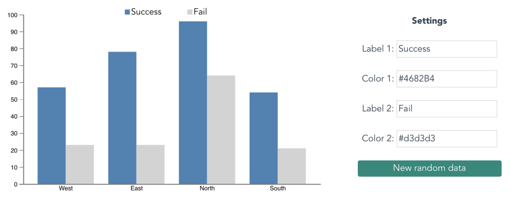
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2017
http://excel.fit.vutbr.cz

UXgraph - Vue.js library with predefined D3 graphs

Olena Pastushenko*



Settings

Label 1: Success

Color 1: #4682B4

Label 2: Fail

Color 2: #d3d3d3

New random data

Abstract
The goal of this project is to create a new JavaScript library, which would contain reusable components for graphs creation. The main challenge is not only to visualize the data on a graph, but also to increase UX of interaction with a final product. And on the other hand - to allow programmers to add new widgets quickly and with minimum input needed. The theoretical part of the project is about defining correct data visualization principles, based on how human brain perceives information. Such principles are fundamental and do not rely on some web development trends, that's why they should be always followed. Most of the existing tools for manipulating with graphs or dashboards offer too many options, and so it is difficult to achieve the best possible representation. UXgraph is a Vue.js library which solves the problem by creating a set of predefined components, which already have default settings. So a programmer needs to connect a dataset, and specify properties only when relevant. At the current stage, several graphs are implemented as independent components and published on a GitHub as an open-source project. More graphs will be added at the next stage. This paper also describes a real life usage example of the library, as the part of the web application for dashboards construction.

Keywords: UX graphs — Data visualization — Vue.js / D3.js library — Informational Dashboards

Supplementary Material: GitHub repository — Live demo

*xpastu01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The focus of this work is to create reusable Vue.js components based on D3.js SVG graphs. The main value is that those graphs are designed in accordance with the best *UX (User Experience)* principles. Those principles are mainly based on the work of Stephen Few [1] and are covered in Section 2. UX defines what impression (experience) users have from using an interface, and how easily they reach thier goals. Components created during this project are gathered into Vue library, which is hosted on GitHub as an open-source tool[1] under MIT License. Beta-version is already published as *npm* (Node Project Manager) library so it is available for all Node.js based projects. There are several angles in which practical utility of this project may be proven. First of all, D3 helps to create a huge variety of different data visualization types [2]. But, the question is how usable and understandable they are. Since D3 offers a lot of possibilities, a programmer may be focused more on finding an appropriate coding solution, then on designing an understandable graph. So, *UXgraph* solves this problem by providing already predefined and easy to use components. The user just needs to connect a dataset and select a color (though default color scheme is provided too). Vue was selected as a base library for this project because it

[1]see Supplementary Materials

is fast, reactive, and provides an opportunity to create new reusable components which can be imported and used as a new custom HTML tag throughout the web page.

Currently, there already exist several Vue.js / D3.js open-source components available on the internet. But firstly, they do not represent even minimum set of needed graph types and allow you to create only one or two types. For example, in Tyrone Tudenhope GitHub project[2] provides only sparklines and line charts. There are a lot of different D3 examples created by Mike Bostock[3], but they do not offer a possibility to quickly set custom settings. The other problem of their components (and the main problem solved in this project) is that they do not fulfill design requirements specified by different graphic visualization experts, Stephen Few [1] in particular. Not following these requirements makes these graphs difficult to understand, and as a result - less effective.

At the current stage, UXgraph also contains only several types of graphs, but they all already follow required design principles and allow easy definition of custom properties, so the first part of the challenge was solved. At the next stage, during the work on my Thesis, all other needed graphs would be added.

Another benefit of this solution is that it already has practical usage example - the web application for Information Dashboard design (Sec. 4). The main motivation for developing that application is to create not just another tool for working with dashboards but to provide a user with a predefined set of widgets, which already satisfy all UX requirements. So, UXgraph helps to achieve that.

2. Ergonomics of the interaction

The big mistake of any software development process is to focus only on the features development, not carrying in mind the general application architecture and vision. At the same time, it is important not to end up developing a business software product which is more entartaining then useful. Cooper calls such effect *dancing bearware* [3]. To build complete and high usability dashboard, it is also important to take into account how people perceive and think. Science that deals with designing things so that they fit the people who use them is called *Ergonomics*. Main ergonomics principles which are relevant to graphs are going to be used for this project and are listed in this Section. Some of them are combined in so-called *Gestalt psychology* [4], which explain rules of clustering perceived items into groups. (Figure 1).

[2]https://github.com/johnnynotsolucky/samples/tree/master/vuejs-d3

[3]https://bl.ocks.org/mbostock

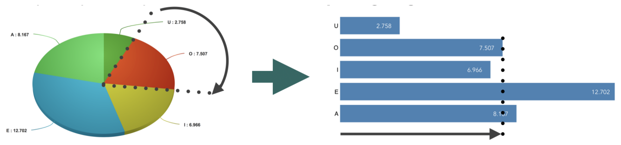


Figure 1. Pie charts are used quite commonly, but they are less effective than the bar charts. Whereas a bar chart uses the line length to encode the quantitive information, pie charts use two-dimensional areas of the slices and their angles, and our visual perception experience troubles comparing angles and 2-D areas.

Besides pre-attentive and attentive attributes shown in the Gestalt Psychology, there are other factors which may influence user's interaction with a software: their experience, goals, and context. Another measure which can be applied to a graph is a concept of *data-ink ratio*, introduced by Edward R. Tufte [5]. Due to this concept, *data-information* is when the ink, used to print a quantitive data, is changing as the data change. Since this project is about software, *ink* may be replaced by *pixels*, but the formula would be the same, and the aim is to make this ratio equal 1:

$$data\,pixels\,ratio = \frac{total\,pixels\,used\,for\,the\,graph}{data\,pixels} \quad (1)$$

So, a very first step in designing correct graphs set was the analysis of possible ways how to improve this ratio. To do this one should firstly reduce non-data pixels, and then enhance the data ones. Figure 2 shows how grid lines in a graph may be eliminated without any information loss.

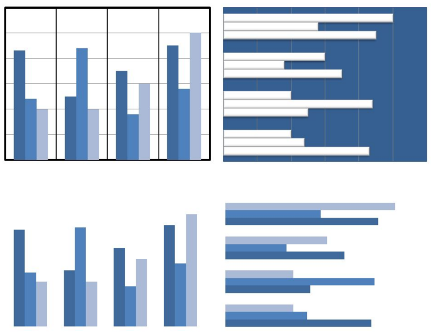


Figure 2. Effect of grid lines elimination.

3. Library implementation

UXgraph requires having a running Vue application or vue.js script connected to any other codebase. *Vue.js* is a *progressive framework* for building user interfaces. It is focused on *the View Layer*, and provides reactive and composable view components. Vue advantage is that even though it is a JavaScript framework, it still supports HTML and CSS right in the single file components (on the contrary of React, where everything is just JS). Before selecting *D3.js* there was performed an analysis of existing JS tools for graphs creation. Following criteria were applied: price, availability of all needed elements, adaptivity. D3 was selected as the only one satisfying all requirements. Another its advantage is that it is highly scalable, so other graphs types may be added to the library in the future.

UXgraph supports latest versions of Vue and D3, and this feature is quite important since with the last version change there were some core updates in both of them[4] [5].

3.1 Display media chosen for the library

It is important to use an appropriate and well-designed display media to reach the full potential of data visualization. Selection of the graph types for this project was based on the following criteria:

- It must be able to efficiently show information when displayed on a small screen.
- It must be the beast mean to display the most commonly used in the dashboards information.
- It must allow creating new graphs with minimum efforts.

According to the main design principles (Sec. 2), the best way to provide information is to display quantitive data in the form of a 2-D graph with X and Y axes. For this project, seven types of graphs were selected, based on their usability. They were preferred because they can provide information more effectively [1] and at the same time - are quite easy to use comparing to the others, which is an important feature both while development and future usage.

Following list shows selected graphs: *bullet graphs*, developed by Stephen Few specifically for dashboards following the purpose to replace gauges and meters[1]; *bar graphs* (horizontal and vertical), used to display multiple instances of one or more key measures; *stacked bar graphs* (horizontal and vertical), which should be used when you need to show several instances of a whole, but with the emphasis on a whole itself; *line graphs*, which are the best ones in showing the shape of data movement and the dynamic and *sparklines*, which were invented by Edward R. Tufte [5] and provide condensed forms of data display.

[4]Vue2.x migration guide
[5]D3v.4 changes description

3.2 Vue reusable components

Components in Vue can help to extend basic HTML elements to encapsulate reusable code. To make it work, Vue's compiler attaches special behavior to these custom elements. To register a global UXgraph component following steps are needed:

```
Vue.component('my-component', {
  // options
})
```

After that, the component may be used in the web page template as a custom element:

```
<my-component></my-component>
```

Every graph type component is located in its own single file template, which contains HTML, scripts for declaring properties and behavior and styling. That makes it possible to use them all independently. Following components are already implemented and can be imported: *Sparklines*, *Linecharts*, *Barcharts*, *HorizontalBarcharts*. *StackedBarcharts* and *BulletGraphs* components would be added during the next stage.

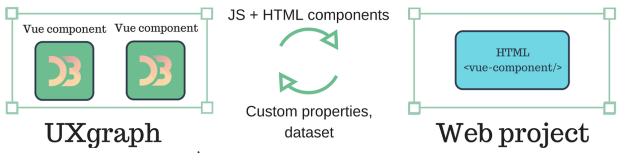


Figure 3. Integration of UXgraph into a Web project

General D3 methods are connected to *.vue* components by installing d3 via *npm*, and then importing *d3* as a global variable. But to make application reactive, it is not enough to create D3 SVG elements as usual. That's why UXgraph uses Vue *mounted* event to call a method for creating a graph after a component instance was rendered. When calling the method it is also needed to specify its parameters, which represent customizable graph settings. For example, for *sparklines* it would be:

```
mounted () {
    this.createSparkline
    ('#id', this.data, this.label,
    this.circle, this.color)
},
methods: {
    createSparkline(id, data, label,
    circle, color) {}
}
```

As a result, variables represented by these parameters may be used anywhere during the SVG construction.

3.3 Passing data between components

Another Vue feature which is used in UXgraph is a possibility to pass data to a component with properties. Since components are reusable and can be inserted in basically any place of a web page, it is important to keep them in their own isolated scope, and so - not to directly reference parent data from a child component. That's why all data in this library are passed to child components using *props*. Firstly, in all child components (which contain graph templates) properties are defined using the *props* option:

```
Vue.component('sparkline', {
  // props declaration
  props: ['data', 'color',
  'circle', 'label']
})
```

After this within a component every property can be reffered as *this.propertyName*. Then they can be passed from the parent template like following:

```
<sparkline
    color="#4682B4"
    label="Daily defects"
    circle=true>
</sparkline>
```

Result is shown on Figure 4.

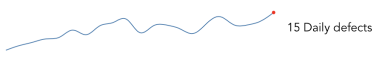


Figure 4. Example of generated sparkline component, with custom parameters

For all UXgraph components default properties are specified, in order to make including of a new component easy. Users need to specify a property only if they want to use custom values.

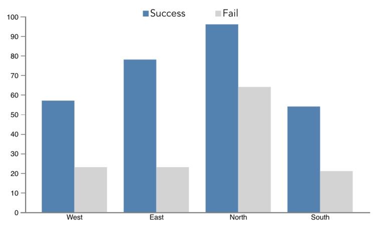


Figure 5. Example of a bar chart component included without any parameters

4. Usage example

A real life example of the UXgraph usage is my Thesis project. The main goal was to create a web application which would allow researchers to easily and quickly create dashboards, all components of which fulfill best UX requirements. App is build using Quasar Framework, which can be later on with the help of Cordova wrapper converted to a hybrid mobile app.

4.1 Primary persona

While working on any application which aims to increase UX, it is important for a programmer to keep in mind the principle "you are not the user". To achieve this, one of the possible ways is to create a primary persona: an archetypal user of a future application. By definition, each primary persona requires a different interface [6]. Since the primary goal of my Thesis is to create an application for internal usage of the research group, there is no need in trying to satisfy the needs of common users. Based on selected target group of users this primary persona was developed: active Ph.D. student who needs to get routine tasks done quickly. Since there is only one primary persona for this project - only one interface is needed.

4.2 Technology stack

The technology stack is a combination of software and programming languages used to create the web or mobile app. There are front-end (client side) and back-end (server side) stacks (Figure 6).

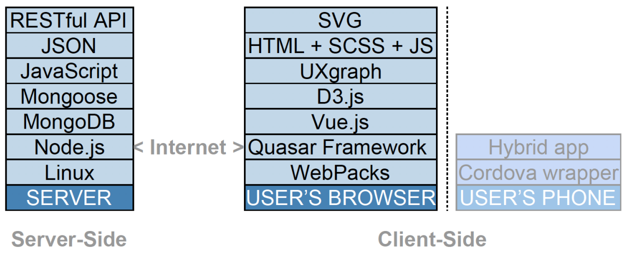


Figure 6. Illustration of technologies used for server and client sides of the application, and explanation how hybrid mobile application may be generated in the next stage.

The base for a front-end of this application is Vue, with connected UXgraph components. Graphs are generated fully on a client side since dataset and other parameters are set as Vue properties. Project uses *Single-Page Application* (SPA) architecture. It loads a single HTML page and dynamically updates it as a user interacts with the app. So, *View Layer* is handled by Vue, with routing managed by *vue-router*. Server communication works with the help of *vue-resource*, plugin for interfacing with a RESTful backend. And everything is built with Webpack build tool.

4.3 RESTful API for handling dashboards operations

An important feature is that all back-end logic is made and hosted as a separate Node.js project, which may be accessed through API. Since front-end and back-end are not closely integrated together, it is possible to add or modify backend operations, without influencing front-end and vice verse.

MongoDB is selected since it works with objects saved as JSON, what makes it easy to export them later on [7]. Plus, it is possible to save objects from *gridstack.js* (used to enable drag-n-drop for widgets construction pane) directly as Mongo JSON objects. All collaboration with MongoDB is managed with the help of Mongoose. This is a MongoDB object modeling tool designed to work in an asynchronous environment.

4.4 Dynamic props

One of the tasks was to update graphs dynamically, whenever their settings are changed. For this, another Vue function is used: *v-bind*. It allows to dynamically bind props to certain data on a parent. So, when the user selects a new color for a graph, this parameter dynamically flows down to the child and it is updated on the graph without page reload. Example usage is:

```
<input v-model="parentColor">
<sparkline
    data="[3,2,1,4]"
    v-bind:color="parentColor">
</sparkline>
```

To redraw graphs when a property is changed without page reload, there were developed custom *watch* functions. Every watcher is triggered when a specific property is changed, and then it calls a method which is responsible for drawing the SVG.

5. Conclusions

This paper describes a basic implementation of a new Vue extension, which uses D3 to create reusable components for data visualization. The main advantage of UXgraph is that it provides a user with a set of predefined graphs, which can be then easily integrated into any Vue or other web development project.

As the first step of a project, the theoretical analysis was made in order to define exact design requirements and minimum needed set of graphs. Several graphs (sparklines, line charts and bar charts) are already implemented as single file library components. Other types of graphs (stacked bar charts and bullet graphs) would be added during the next stage, in terms of my Thesis project.

The main challenge was not to provide a user with as many options as possible, but give them an instrument for creating an efficient and easy to work with graph or dashboard.

Library is already published as *npm* module. And even since it currently has only several types of graphs, and functionality is limited (it is still in beta-version), it already had more then 850 downloads during 1 week only. This shows high demand on such a library.

This paper shows one example of UXgraph usage - a web application for dashboards construction. But an advantage of having each graph as a separate independent component in a library is that it may be used not only for complicated dashboards but on any web page.

Since it is open-source and available on GitHub, any developer can contribute or include it in their project. Further library development and support are planned.

Acknowledgements

I would like to express my gratitude to my supervisor Jiri Hynek for his collaboration, useful comments, remarks, borrowing books and engagement during this project. And also for always having time for a consultation and discussion. Furthermore, I would like to thank Vero Vanden Abeele, Groep T, KU Leuven for introducing me to the topic of Human-Computer Interaction (HCI) as well for the book's recommendations.

References

[1] Stephen Few. *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Media, 2006. ISBN: 0596100167.

[2] Malcolm Maclean. D3 tips and tricks v4.x. leanpub.com, Dec 2016. https://leanpub.com/d3-t-and-t-v4.

[3] Cooper. *The Inmates Are Running the Asylum*. Indianapolis, 1999.

[4] Jeff Johnson. *Designing with the Mind in Mind, Second Edition: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann, 2014. ISBN: 0124079148.

[5] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2001. ISBN: 1616890584.

[6] Jesse James Garrett. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders, 2010. ISBN: 0321683684.

[7] Simon Holmes. *Getting MEAN with Mongo, Express, Angular, and Node*. Manning Publications, 2015. ISBN: 1617292036.