INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 4, Issue 2)
Available online at: www.ijariit.com

A visualization engine – Generating reusable and shareable visual components and dashboards

Niladri Bihari Mohanty
niladri.mohanty@gov.in
National Informatics Centre, Bhubaneswar, Odisha

Snehasish Nayak
sn3hasishnayak@gmail.com
National Informatics Centre, Bhubaneswar, Odisha

Pratibha Singh
pratibha.singh@nic.in
National Informatics Centre, Bhubaneswar, Odisha

Ashok Kumar Hota
ak.hota@nic.in
National Informatics Centre, Bhubaneswar, Odisha

ABSTRACT

Aesthetically designed high-impact dashboard is a challenge for any project. Development of dashboard needs focused and thoughtful visualization interface. However the effectiveness of the dashboard depends upon the selection of the data in the specific context. Therefore it is important to give more focus on the core concerns like data extraction, transformation, loading and analytics. The visualization Engine presented in this paper automates the process of visualization. It eliminates the need of any program to generate visualizations. Hence drastically minimize the code needed to be written to use or share the visualizations in a very secured, optimized and managed way. The paper outlines the process to assemble many visual components into a dashboard and use or share the same dashboard as a whole in any portal.

Keywords: *Graph, dashboard, visualization, analytics, Fusion Chart, D3.js, Java Script based visualization, Responsive, reactive graphs, visualization tool.*

1. INTRODUCTION

In the era of SMAC (Social Mobile Analytics Cloud) enabled application development, analytics is playing a pivotal role in any project. Human sensors have created and are creating tons and tons of data. Insights of these data are represented through visualizations in a meaningful context in different projects. This process involves mathematics, statistics, predictive modeling and machine-learning techniques to find meaningful patterns and knowledge, existing in recorded data which is very challenging. This study is focused to develop the visualization of the insight in a very simple, code free, loosely coupled and centrally but distributed manner. Hence the development time could be utilized in the real brain storming activities to get those insights rather than putting effort on the visualizations.

Efforts to simplify visualization are not new and many libraries, frameworks, techniques have been practiced in the last couple of years. But all these techniques are either more complex than the problem it is trying to solve or exposes new problems or issues. Methods implemented in majority of such scenarios are sharing visualization using URL over the iFrame. But iFrame is not considered as secured. It creates problems in getting responsiveness in the design too. Cross Origin Service Workers based on foreign fetch is also difficult to implement in the case of iFrame.

The solution discussed here has addressed both concerns.

- Generating visualization components through metadata
- Use as well as share those components in a very secured, optimized and managed way.

It has also defined process, through which the native look and feel of the component in the consumer portal can be maintained. It further outlines about achieving the offline friendly experience through the use of service worker. The problem of isolating the visualization rendering technology from its implementation has also been addressed in a more managed way.

2. RELATED WORK

Many dashboard building solutions have sprung up to meet the demand. Mostly those are based on the techniques of GUI enabled metadata capture. But most of them don't have any sharing technologies. Still there are few who provide GUI based visualization rendering and also provide technologies to share those visualizations to different domains. One of the most popular site data.gov.in has a utility named Visualization Engine Version 3.01 where visualization components are created by entering metadata through a GUI but the sharing technologies is totally based on inline frame of HTML or IFrame. During study it was observed that this methodology of using inline frame is very common approach for the same purpose and is being used in almost all the occasions. In fact the practice is so common in nature that Google shares or embed it's map and YouTube videos through iFrame only which was first introduced by Microsoft in the year 1997. Clickjacking2, Cross Frame Scripting and iFrame Phishing3, 4 kinds of terms are not new to the world of both security and hacking. In many organizations use of iFrame is also considered as a point of vulnerability in their security development life cycle. At the same time in the era of responsive web development, iFrame lays many obstacles to achieve responsiveness without compromising the security aspect of the application. Keeping eyes on the various issues over the use of iFrame, W3C has added a new set of features to the HTML termed as Web Components5 to improve creation of reusable widgets and components. This newly added module has four great features, out of which 'HTML Imports' is our center of attraction. During our study it was found that although the concept is very useful and relevant to resolve our purpose but the objective is very generic to import HTML in another HTML. This specific feature is still not supported by majority of the browsers including Mozilla Firefox. Again by importing one html page it creates head and body within the body tag of the parent DOM.

Features				
HTML Import	✓	✗	✗	✗

Fig 1: Status of browsers supporting HTML Import

Ex. If the content of the head tag is importing the JQuery.Js file then for 'n' number of visualizations, the same JQuery will be loaded 'n' numbers of times. To avoid this problem a complete new but very simple approach has been attempted in the paper.

3. INNOVATION

In the most conventional approach, the developers are used to write codes responsible to render the visualization in the browser side. The browser connects the data source through Rest API to get the respective data for the visualization. After release of HTML5 as W3C recommendation, the new concept of 'custom attribute' brought many new possibilities to their final shape. The data-* is now capable to store data in the HTML tag which can be further handled by any JavaScript.

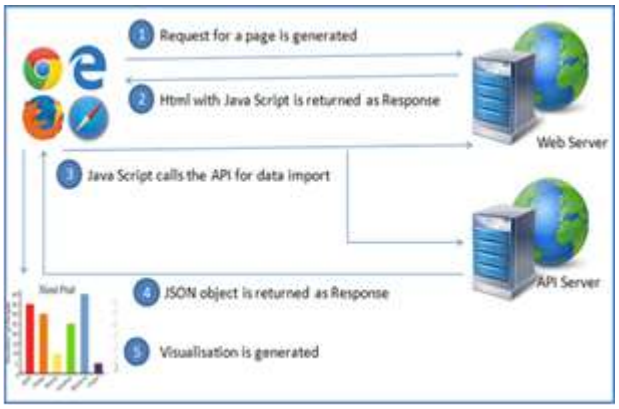

Fig 2: Conventional method to render visualization

The release of AngularJS by the team of Google has extensively used this trick in the form of angular directives.

Example: data-ng-App= "MyApp" where ng-App is the attribute and "MyApp" is the value to the attribute. The browser understands, that this is a custom attribute as it is prefixed by data-*. The attribute is handled by the JS file named angular.min.js.

The similar line of thought has been drawn in our approach for visualization sharing policy.

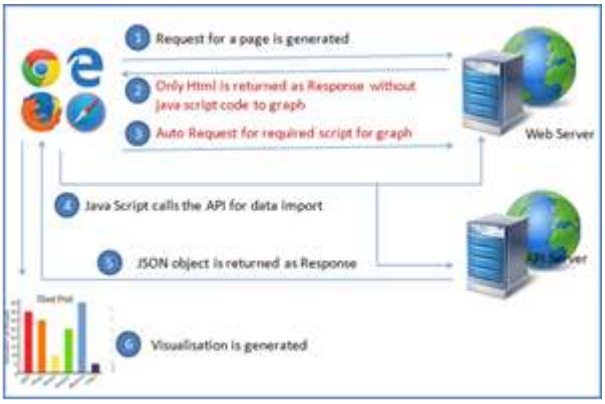

Fig 3: Info flow with proposed visualization Engine

The source organization will enter the metadata of the visualization in the GUI based web interface. These metadata contains the information like the type of visualization, Title, URL of the API through which data need to be grabbed and other similar information. Upon completion of this process the visualization is generated for preview and a unique ID number gets assigned to the visualization.

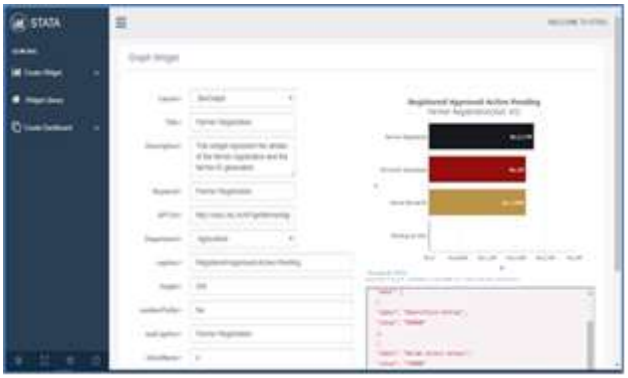

Fig 4: Metadata capturing GUI - one time entry for each visualization

This particular graph can be embedded inside any html page using the HTML based custom attribute predefined as data-nic= "<visualization_id>". Now the question is how the browser will understand in which way to treat the custom attributes? The NICDB.js which is referenced in the head tag facilitates this. Assuming the visualization id as "OSS1", the final code would be very simple as mentioned below.

```
<html>
<head>
<script src="URL/nicdb.js"></script>
</head>
<body>
<div id="A" data-nic="OSS1"></div>
</body>
</html>
```

Further the visualization can be a single component or entire dashboard. When we have a library of huge visualization components, creation of a dashboard is just a matter designing the layout and assembling the related and required components. For the same purpose a layout designer GUI has also been developed as below. Bootstrap based responsive layout can be dragged and draw using the proposed utility.

Fig 5: Dashboard Layout designer

Once the layout of the dashboard is ready to give space for various visualization components of different sizes, then the drag & drop is the only action need to be carried out to create the dashboard.

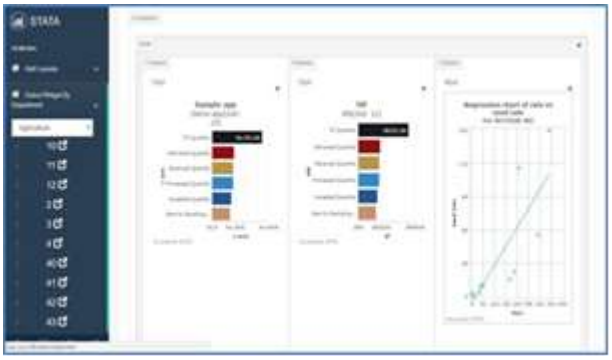

Fig 6: Dashboard designed by dragging from left panel

Once the dashboard has been designed and saved, a unique ID number is assigned to the dashboard which can be used latter to share or embed the entire dashboard to any web pages.

```
<html>
<head>
<script src="URL/loadDB.js"></script>
</head>
<body>
<div id="A" data-nicdb="40"></div>
</body>
</html>
```

4. ARCHITECTURE

Whiling keeping the usability of the system simple it has been divided in to three main components.

4.1 Metadata Capturing Module

This module is responsible to capture the metadata of the visualization. It may be noted that the data to be presented is not the part of it. The API to fetch the data from the source is only stored as part of the metadata. This removes the complexity of storing source data in middle layer. Middle layer stores only the URL to the source data.

4.2 Visualization Code Generator

This module is responsible to take the 'id' of the graph from the browser end. Along with the metadata retrieved from the Database for that respective visualization, the code generator generates the required Java Script code to render the visualization in the client end.

4.3 Client Module

This module is playing the vital role in the user end for cross checking the availability of required ecosystem. If it is not found then the module automatically downloads those before loading the visualization rendering script generated by the "Visualization Code Generator" module. This is also responsible to handle the custom HTML attribute data-nic=*.

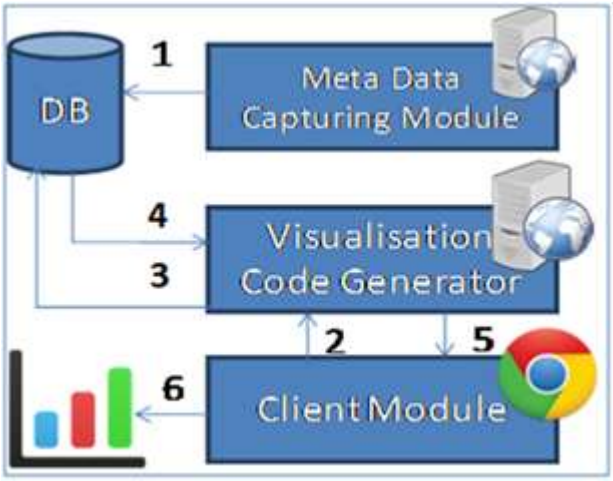

Fig 7: Workflow of the Visualization Engine

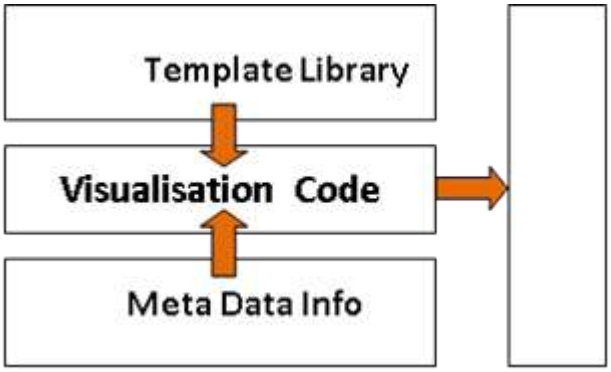

Fig 8: Generation of Visualization Code

5. SECURITY CHALLENGES

While using the visualization engine, as the user loads scripts from an external server, there is a possibility that, someday that external server can get hacked and the script can get replaced by some malicious codes. If this happens, all the websites using our visualization engine to render visualizations can get affected

These types of problems are not new. There are solutions available for it and currently, all the available content delivery networks (CDN) are using this workaround. The solution is called Sub Resource Integrity (SRI).

The idea of Sub Resource Integrity is to include the script along with its cryptographic hash (e.g. SHA-384) when creating the web page. The browser can then download the script and compute the hash over the downloaded file. The script will only be executed if both hashes match.

```
<script src="URL/nicdb.js" integrity="sha384-
qVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7
HNQlGYl1kPzQho1wx4JwY8wC"></script>
```

The first part of the integrity tag says about the Hashing algorithm used to generate the digest so that the browser will use the same algorithm to validate it.

The security properties of a collision-resistant hash function ensure that any modification results in a very different hash. This helps the site owner to detect and prevent any changes, whether they come from a compromised source.

6. FUTURE SCOPE

It is planned to implement few major additions to our visualization engine in near future. Some of them are mentioned below:

- The UI/UX of the platform needs to be improved to make more robust, extensible and easy to use.
- As the engine is working mostly on JSON data so it is planned to change the platform to MEAN stack.
- Migration to D3.JS based visualization has been envisioned.
- Maximum optimization through the help of various caching methodology.

- Implementation of Service worker to provide highly improved offline experience.

As we are collecting the metadata from different visualizations, we have planned to create cataloguing based on those metadata like department name, Keywords, type of visualization etc., making the visualizations more searchable and easy to discover.

7. CONCLUSION

With the approach to develop visualization in a holistic way keeping a union of utility, usability and simplicity, the engine facilitates design of focused, thoughtful, and user-friendly dashboards. The architectural design based on in-depth study, has enabled development of the secured, scalable visualization engine which generates reusable and sharable visual components and dashboards. With this effective tool at hand developers can focus on delivery channel, level of interactivity, timeliness of data, and analytical capabilities.

8. REFERENCES

[1] Visualization Engine 3.0 https://visualize.data.gov.in/
[2] Click jacking: Attacks and Defenses by Lin-Shung Huang Carnegie Mellon University linshung.huang@sv.cmu.edu Alex Moshchuk Microsoft Research alexmos@microsoft.com Helen J. Wang Microsoft Research helenw@microsoft.com Stuart Schechter Microsoft Research, stuart.schechter@microsoft.com Collin Jackson Carnegie Mellon University collin.jackson@sv.cmu.edu
[3] https://www.owasp.org/index.php/Cross_Frame_Scripting
[4]http://resources.infosecinstitute.com/iframe-security-risk/.

