H2020 Project: Smart Resilience Indicators for Smart Critical Infrastructure
D4.3 - Design and application of interactive visualization for RIs

Coordinator: Aleksandar Jovanovic EU-VRi
Project Manager: Bastien Caillard EU-VRi
European Virtual Institute for Integrated Risk Management
Haus der Wirtschaft, Willi-Bleicher-Straße 19, 70174 Stuttgart
Contact: smartResilience-CORE@eu-vri.eu
Design and application of interactive visualization for RIs

Report Title: Design and application of interactive visualization for RIs

Author(s): I. Shapira, U. Barzelay

Responsible Project Partner: IBM

Contributing Project Partners:

<table>
<thead>
<tr>
<th>Document data:</th>
<th>File name / Release:</th>
<th>D4.3_Visualization_Design_v06bc30042018</th>
<th>Release No.: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages:</td>
<td></td>
<td>35</td>
<td>No. of annexes: 2</td>
</tr>
<tr>
<td>Status:</td>
<td></td>
<td>Final</td>
<td>Dissemination level: PU</td>
</tr>
</tbody>
</table>

Project title: SmartResilience: Smart Resilience Indicators for Smart Critical Infrastructures

Grant Agreement No.: 700621

Project No.: 12135

WP title: WP4. Defining classic and deriving smart Resilience Indicators (RIs)

Deliverable No.: D4.3

Date: Due date: April 30, 2018

Submission date: April 30, 2018

Keywords: Resilience, interactive data visualization, resilience indicators

Reviewed by: P. Klimek

Review date: April 13, 2018

L. Bodsberg

Review date: April 16, 2018

Approved by Coordinator: A. Jovanovic

Approval date: April 29, 2018

Haifa, April 2018
## Release History

<table>
<thead>
<tr>
<th>Release No.</th>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mar 29, 2018</td>
<td>1st Draft</td>
</tr>
<tr>
<td>2</td>
<td>April 29, 2018</td>
<td>Final version</td>
</tr>
</tbody>
</table>
Modern critical infrastructures are becoming increasingly smarter (e.g. the smart cities). Making the infrastructures smarter usually means making them smarter in the normal operation and use: more adaptive, more intelligent etc. But will these smart critical infrastructures (SCIs) behave smartly and be smartly resilient also when exposed to extreme threats, such as extreme weather disasters or terrorist attacks? If making existing infrastructure smarter is achieved by making it more complex, would it also make it more vulnerable? Would this affect resilience of an SCI as its ability to anticipate, prepare for, adapt and withstand, respond to, and recover? What are the resilience indicators (RIs) which one has to look at?

These are the main questions tackled by SmartResilience project.

The project envisages answering the above questions in several steps:

1. By identifying existing indicators suitable for assessing resilience of SCIs (RI1)
2. By identifying new smart resilience indicators including those from Big Data (RI2)
3. By developing a new advanced resilience assessment methodology based on smart RIs and the resilience indicators cube, including the resilience matrix (RI3)
4. By developing the interactive SCI Dashboard tool (RI4)
5. By applying the methodology/tools in 8 case studies, integrated under one virtual, smart-city-like, European case study. The SCIs considered (in 8 European countries!) deal with energy, transportation, health, and water.

This approach will allow benchmarking the best-practice solutions and identifying the early warnings, improving resilience of SCIs against new threats and cascading and ripple effects. The benefits/savings to be achieved by the project will be assessed by the reinsurance company participant. The consortium involves seven leading end-users/industries in the area, seven leading research organizations, supported by academia and lead by a dedicated European organization. External world leading resilience experts will be included in the Advisory Board.
Executive Summary

This report describes the design of the interactive visualization for resilience indicators web application. This web application design is based on the methodology for assessing, predicting, and monitoring the resilience of SCIs described in T3.5 and is aimed to demonstrate the feasibility of constructing an interactive data visualization with a possible manifestation. This report describes in detail the basic concept of web applications (a client–server computer program in which the client runs in a web browser) and how it is implemented in the SmartResilience interactive visualization for resilience indicators. It describes the expected JSON input data format which is generated from the data in the SmartResilience database. This report describes various aspects of the web application: web technologies applied (such as nodejs, D3), how to use it (input data, buttons, clicks), and how to get insights about a use case from it (meaning of colors, type of visualization, switch between use cases).
Table of Contents

Release History ...........................................................................................................ii
Executive Summary .....................................................................................................iv
Table of Contents .......................................................................................................v
List of Figures ................................................................................................................vi
List of Acronyms .........................................................................................................vii

1 Visualization design .................................................................................................8
  1.1 Background for web application ...........................................................................8
  1.2 Our web application ............................................................................................8
    1.2.1 Server-side - nodejs ..................................................................................9
    1.2.2 Client-side – D3 .....................................................................................10
      1.2.2.1 Input data – JSON ..........................................................................10
      1.2.2.2 OpenStreetMap .............................................................................13
      1.2.2.3 D3 library .......................................................................................15
      1.2.2.4 Treemap – D3 layout .....................................................................15
    1.2.3 Functionality ..............................................................................................18

2 Conclusions ................................................................................................................25

References ..................................................................................................................26

ANNEXES ..................................................................................................................27

All items must be labeled “H2020 SmartResilience” so as to be able to identify them in the event of a divestiture.
List of Figures

Figure 1: Our high-level web application design ........................................................... 9
Figure 2: The basic “Node” structure ........................................................................... 10
Figure 3: Example of root “Node” and one child structure ........................................... 11
Figure 4: Example of “Node” and two child structures .............................................. 12
Figure 5: Example of “Node” and three leaves (child structures) .............................. 13
Figure 6: Extended JSON when using the map ......................................................... 14
Figure 7: Example of map JSON ............................................................................. 14
Figure 8: Map with Paris, Brussels, Munich polygons .............................................. 15
Figure 9: Cell size by global weight ........................................................................ 16
Figure 10: Evenly sized cells ................................................................................... 16
Figure 11: Resiliency-level colors ............................................................................ 17
Figure 12: T3.2 level colors ...................................................................................... 17
Figure 13: Ordinal color scales - schemeSet3 ............................................................ 18
Figure 14: Efficient use of space by the number of cells ........................................... 18
Figure 15: The three parts of the visualization web page ........................................... 19
Figure 16: Toggle Map / not using the map ............................................................... 20
Figure 17: Toggle Map / selecting Brussels in the map .............................................. 20
Figure 18: Load Json File / select the JSON file to visualize .................................... 21
Figure 19: Cell color and cell size ............................................................................ 21
Figure 20: Step 1 - Start with global weight visualization ........................................ 22
Figure 21: Step 2 – Change to evenly visualization .................................................. 22
Figure 22: Step 3 – Cells moving to evenly visualization .......................................... 22
Figure 23: Step 4 – Ending with evenly visualization .............................................. 23
Figure 24: Color by resilience/score (default) ......................................................... 23
Figure 25: Color by T3.2 level colors ...................................................................... 23
Figure 26: Color by siblings ................................................................................. 24
Figure 27: Details for indicator ............................................................................... 24
Figure 28: Details for phase ................................................................................... 24
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>CI</td>
<td>Critical Infrastructure</td>
</tr>
<tr>
<td>D</td>
<td>Deliverable</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>D3</td>
<td>Data Driven Documents</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>IE</td>
<td>Internet Explorer</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>NPM</td>
<td>The Node Package Manager</td>
</tr>
<tr>
<td>OSM</td>
<td>OpenStreetMap</td>
</tr>
<tr>
<td>RI</td>
<td>Resilience Indicator</td>
</tr>
<tr>
<td>RIL</td>
<td>Resilience Level</td>
</tr>
<tr>
<td>SCI</td>
<td>Smart Critical Infrastructure</td>
</tr>
<tr>
<td>T</td>
<td>Task</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
1 Visualization design

1.1 Background for web application
A web application is a client–server computer program in which the client runs in a web browser [1].
When developing visualization for a project, we need to consider the delivery platform (to maximize usage flexibility) and select the most adequate development language to save time and increase efficiency.
A web application, using standardized communication protocols and development languages, simplifies developers’ work in supporting different variants of clients, hardware, and operating systems, allowing for faster development and similar functionality across all platforms [2].
The client side of a web application is coded in a browser-supported language, such as JavaScript or HTML. Today, there are many advanced libraries for web development (like the D3 library we use) that boost the development process while enabling advanced features.

Web applications are usually broken into logical chunks called “tiers”, and each tier is assigned a role. The most common structure is the three-tiered application, in which the three tiers are labelled “presentation”, “application”, and “storage”.
The first tier is the web browser, which addresses the presentation of the information (also referred to as the client side), and is written in a browser-supported script language (like HTML or JavaScript). The middle tier is the application engine that uses a dynamic web content technology (such as ASP, CGI, ColdFusion, Dart, JSP/Java, Node.js, PHP, Python, or Ruby on Rails). This tier deals with more complex computing, like storing and retrieving the information (also referred to as the server-side). The third tier is the database (storage). The web browser sends requests to the middle tier, which services them by making queries and updates to the database, and reports the results back to the user interface.

1.2 Our web application
The visualization methodology proposed in D3.5 presents an abstract way of translating indicator-related data to the visual domain of a computer screen, composed of pixels and colors. Based on that visualization approach, we implemented a web application [3] that is one possible manifestation of D3.5, and which presents the feasibility of the D3.5 methodology.
Our application takes as input a use case along with its indicator data, calculates an intermediate score that represents the resilience of the data, and visualizes it in a customized treemap format.
Our goal for the design is to provide the user with interactive graphics, allowing him to explore and derive as much insight from the data as possible, as detailed in D3.5.
Our application takes into account the following requirements:
- Support big JSON files as input
- Datasets may be large
- Flexibility is required to switch between different views of the same data and different use cases
Following state-of-the-art web technologies, we developed our web application in Node.js for the server-side and D3 library for the client-side.

Figure 1 describes a high-level design of our web application architecture. The web server is the engine that enables the user to upload use case input data and, as a response, produce dynamic web page content for the user interface. The web client implements an interactive visualization corresponding to the data and the user settings; it presents buttons to the user configuration that enable the dynamic change of the visualization.

We expect that the use case input data will be in the format described in 1.2.2.1 (a canonical form fit for D3 library). If the input data comes in a different format, some adapter must be applied to transform the data to the expected format before calling the web application.

1.2.1 Server-side - nodejs

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code on the server-side of a web application. Its purpose is to produce dynamic web page content. It is based on event-driven architecture and asynchronous I/O to optimize throughput and scalability in web applications with many input/output operations [7-8].

For security reasons, some browsers won’t allow to load a local file via JavaScripts. This means that the client-side code that loads the input JSON file for visualization will fail.

A simple and more reliable solution to this problem is to load the web page using the web server [16]. The browser sends a GET request for a web page and the web server responds with the web page [22].

On the server side, we used some public packaged modules that made the development faster and easier. To load these modules, Node.js comes with an installation tool named the Node Package Manager (NPM or npm). The npm tool comes with a Node.js installation. It is a set of public-packaged modules available through easy installation via an online repository, with version and dependency management [17].

Here is a list of npm modules we use:

1) express – a fast, minimalist web framework [9]. The Express philosophy is to provide small, robust tooling for HTTP servers, making it a great solution for single page applications, web sites, hybrids, or public HTTP APIs.

2) express-handlebars - a Handlebars view engine for Express, enable to separate the layout and content of a web page [10].

3) body-parser – this parses incoming HTTP request bodies [12]. It makes it easier to get at the information contained in client requests in a variety of formats.

1.2.2 Client-side – D3

In this section, we describe the client-side components. We describe in detail the expected input data format and the D3 library and its treemap layout.

1.2.2.1 Input data – JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types. It is a very common data format used for asynchronous browser–server communication [14].

JSON provides a set of rules to encode general data structures into a well-defined physical form. It’s a simple yet expressive language based on the type system of JavaScript. JSON is able to express rich structural semantics with little space taken by syntactic notations, striking a balance among readability, compactness and expressiveness.[23]

As explained in D3.5, the resilience dataset is composed of dozens and up to hundreds of indicators [3]. Based on the resilience assessment methodology, referred to as CIRAM and defined in D3.2, we map the six levels of assessment into a matching JSON file with tree structure, which can easily be used by D3. We consider each level as the child of the level before it. The leaves are the indicator features that can be formed as attribute-value.

Our web application JSON format

We define a common data structure “Node” that is used to describe one of the six levels of assessment. Using it, we are building a tree structure for the use case data.

![Figure 2: The basic “Node” structure](image)

**Detailed description of each field**

**type** - One of the six levels: Area, CI, Threat, Phase, Issue, and Indicator

**name** - A description of the type
myWeight - The weight of the current node (for the root “Node” this is 1)
minScore - The minimum score this “Node” can get
maxScore - The maximum score this “Node” can get
score – The actual score of this “Node”. This is calculated based on its children scores (sum of all children:
child.score * child.myWeight)
   For Indicator: we normalize the real value to a range of (minScore, maxScore)
   minScore + (maxScore - minScore)*((realValue - minValue) / (maxValue - minValue))
normalizedScore – The normalized score of this “Node” to range of (0,1)
   (score - minScore) / (maxScore - minScore)
minValue - The minimum value this “Node” can get (applicable only for Indicator)
maxValue - The maximum value this “Node” can get (applicable only for Indicator)
realValue - The real value this “Node” can get (applicable only for Indicator)
numChildren – Number of “Node” children
children – Array of “Node” children
childrenWeight - Array of the children myWeight (should sum up to 1)

This visualization depends on the JSON file values, but it is adaptive to the values.
In the following figure examples, we used:
Scores on a scale of [1, 5] - for Issue and Indicator levels
Scores on a scale of [1, 10] – for all other levels

Figure 3: Example of root “Node” and one child structure
Figure 4: Example of “Node” and two child structures
1.2.2.2 **OpenStreetMap**

OpenStreetMap (OSM) is a collaborative project creating a free editable map of the world [15]. It is a very flexible platform with many APIs and services that allow users to create their own map style.

We use OpenStreetMap to visualize the use case by the associated city. We define an extended JSON file to the JSON definition in 1.2.2.1 (see Figure 6) that also includes geographical information of the city for each use case. Using the OpenStreetMap API and the given geographical point in the JSON file, we draw a polygon on the map around a use case city, and when the user clicks on the polygon (city), we fetch its corresponding data (JSON data file) for the visualization. In this way, we allow the user a simple/interactive way to switch between use cases.

To support this, we defined the following JSON file format:
Figure 6: Extended JSON when using the map

Detailed description of each field

- **name**: The use case city name (this name will appear on the popup when hovering over the polygon)
- **polygon**: An array of geographical points with certain latitude and longitude values
- **color**: The color of the polygon
- **data**: The input JSON data as describe in 1.2.2.1

Figure 7: Example of map JSON
D3 library

D3 (Data Driven Documents) is a state-of-the-art web technology. It is a JavaScript library for producing dynamic, user-interactive data visualizations in web browsers.

In D3, the “data” is provided by the user (like a JSON file), the “documents” are web page content (like HTML), and the “driven” is done by D3 by connecting the data to the documents [16].

D3 facilitates generation and manipulation of web documents with data. It does this by:
- Loading data into the browser’s memory
- Binding data to elements within the document, while creating new elements as needed
- Transforming those elements by interpreting each element’s bound datum and setting its visual properties accordingly
- Transitioning elements between states in response to user input

When using D3, the developer sets the visualization rules (like the color palette to use, what to store on the graph x-axis, etc.) and D3 executes it.

Treemap – D3 layout

Treemap displays hierarchical data as a set of nested rectangles. Each branch of the tree is given a rectangle, which is then tiled with smaller rectangles representing sub-branches. A leaf node's rectangle has an area proportional to a specified dimension of the data. Often the leaf nodes are colored to show a separate dimension of the data [4-6].

One of the benefits of the treemap is when the color and size dimensions of a cell are correlated in some way with the tree structure, the user can easily see patterns that would be difficult to spot in other ways. Based on this and on the data that we maintain, we implemented the following rules and design for cell size, cell color, and scaling:
Cell size

The treemap view is composed of cell layers, in which each cell represent a “Node”. We propose two options to the user for determining the cell size to get different insights. Cell size can be determined by global weight, or they can be evenly sized between siblings.

Cells sizes using global weight

Cell size can be determined by global weight, which is calculated as multiple myWeight (the “Node” weight) from the root “Node” till (including) this current “Node”.

$$0.08038 = 1 \times 1 \times 0.5455 \times 0.4 \times 1 \times 0.3684$$

- The global weight of Indicator_1 is 8.038%, meaning, this indicator determines 8% of the Level-1 (e.g., Tel-Aviv area) resilience level is
- The cell of Indicator_1 will take 8% of the area

Figure 9: Cell size by global weight

Evenly sized cell per level

A cell size can be determined by evenly dividing its parent size between the number of its children.

Figure 10: Evenly sized cells
Cell color
The treemap view is composed of cell layers, in which each cell represent a “Node”. We propose three options for the user to determine the cell color to get different insights. A cell color can be determined by T3.2 level colors, resilience/score, or siblings.

Color by score and resiliency level
Since the goal of our visualization is to visualize the resiliency level and gain insight from it, in this option we color each cell according to its resiliency score. We define five score levels [Excellent, Good, Average, Poor, Critical] as follows:

![Figure 11: Resiliency-level colors](image)

Color by T3.2 level colors
In Task 3.2, we define six levels of assessment (Area, CI, Threat, Phase, Issue, and Indicator). This option colors each cell according to its assessment level as follows:

![Figure 12: T3.2 level colors](image)

Color by sibling
In a tree structure, we define the notion of siblings as children “Nodes” with the same parent. In this option, we color all sibling cells with the same color.
Scaling and setting order within a group of indicators

Treemaps are advantageous, among other reasons, because they make efficient use of space and can represent a large number of elements. To make the visualization clearer for each level, we sort the elements according to their resiliency score.

**Figure 13: Ordinal color scales - schemeSet3**

**Figure 14: Efficient use of space by the number of cells**

**1.2.3 Functionality**

**Server-side**

As mentioned in 1.2.2.3, the server-side is implemented in Node.js (written in JavaScript). We implemented an HTTP server using the “express” library.
The server loads the given input data JSON file and creates a matching web page using D3 (this is to overcome the security restriction describe in 3.2.1)

**Client-side**

Our visualization implementation divides the HTML page into three parts: Loading input data to visualize, the treemap dynamic visualization, and the details of the selected Node. In the following sections, we describe the functionality of each part.

![Image of the three parts of the visualization web page](image)

**Figure 15: The three parts of the visualization web page**

1) **Loading input data to visualize**

We offer the user two options for loading the input data, using either the **Toggle Map** or **Load Json File** buttons.

- **By selecting Toggle Map**, users can toggle between using or not using a map. When selecting to use a map, OpenStreetMap is visualized and loaded with the matching JSON file data. Based on the data, the map appears with polygons around those cities participating in the use case. Clicking on one of those cities loads the matching JSON file with the corresponding data.
- **Selecting Load Json File** allows users to load any JSON file to view. Clicking on the **Load Json File** button opens a file explorer where the user can select the JSON file to load.

Figure 16: Toggle Map / not using the map

Figure 17: Toggle Map / selecting Brussels in the map
2) Treemap dynamic visualization

As mentioned in 1.2.2.3 and 1.2.2.4, we are using the D3 library with a treemap layout to visualize the data. We offer the user an interactive option (using check boxes) to change two visualization rules (cell size and cell color) and an option to dynamically change the visualization on the same data.

Determining the cell size (described in 1.2.2.4)

In the following figures, we show the dynamic visualization changes when switching from cell size by global weight to cell size determined evenly.

The matching JSON file used for the following figures can be found in Annex A.2.1.
Figure 20: Step 1 - Start with global weight visualization

Figure 21: Step 2 – Change to evenly visualization

Figure 22: Step 3 – Cells moving to evenly visualization
Determine the cell color (T3.2 level colors, resilience/score, or siblings)

In the following figures, we show the different cell colors. The matching JSON file used for the following figures can be found in Annex A.2.1.

Figure 23: Step 4 – Ending with evenly visualization

Figure 24: Color by resilience/score (default)

Figure 25: Color by T3.2 level colors
3) Details of selected Node

In some cases, the user wants to view the details of a specific cell to get insight. Users can hover over a specific cell (“Node”) to get a display with detailed information on the “Node”.

**Figure 26: Color by siblings**

**Figure 27: Details for indicator**

**Figure 28: Details for phase**
2 Conclusions

In this report, we describe the design of the “interactive visualization for resilience” web application. We start by describing the basic concepts of web applications (such as the three-tiered application). We then describe how we divided our web application into two parts: server-side, which loads the given input data JSON file and creates a matching web page, and client-side, which provides an interactive visualization.

Using details and examples, we describe the expected input data tree structure (JSON format); we show how the tree structure is based on the resilience assessment methodology (CIRAM) and can easily be used by a D3 treemap layout (by mapping “Node” in the tree structure to “cell” in the D3 treemap layout). We describe the meaning of the score and weight fields, show how each Node score is based on its children’s score and weight, and explain how it is reflected in the visualization.

We show how we create a dynamic interactive visualization by (i) switching between cell size types (global weight or evenly size between siblings), (ii) switching between cell colors (T3.2 level colors, resilience/score, or siblings), and (iii) switching between use cases (loading a JSON file or using the map). Using examples, we show how different visualizations on the same data (like changing colors or cell sizes) can provide insights about the data. We show that even though we are visualizing complex data, we don’t lose data, and if required we can get all the detailed information.
References

[3] SmartResilience (2016). Deliverable D3.5: Interactive Visualization as support to indicator-based decision making
ANNEXES

Annex 1: Review process
Annex 2: JSON sample
Annex 1  Review process

The Content of this Annex has been submitted as part of the periodic review report to the PO/EU/ Reviewers.
SmartResilience: Indicators for Smart Critical Infrastructures

Page 34