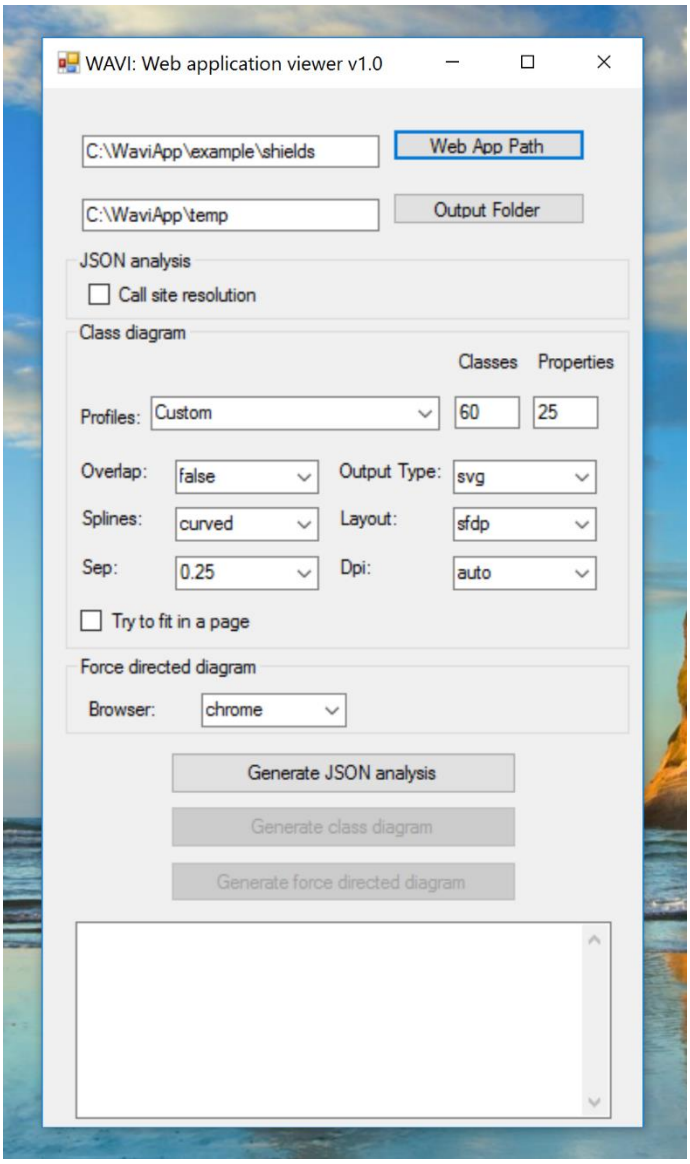
WAVI STANDALONE - QUICK MANUAL



SYSTEM REQUIREMENTS

- Windows 7/8/10 64bit
- Microsoft .NET Framework 4.5.2 or better

INSTALLATION

1. This version of wavi is standalone, copy the files the desired directory.
2. Execute “app.exe”

GENERATE JSON ANALYSIS (TEXTUAL REPRESENTATION)

1. Select the path to the web application you want to reverse engineer.
2. Select the output folder (where textual and graphical representation will be saved).
3. Choose if you want to include call site resolution (link between function call site and function declaration) by checking the « call site resolution » checkbox.
4. Click on Generate JSON analysis.

GENERATE FORCE DIRECTED DIAGRAM

1. First you need to generate the textual representation.
2. You can choose between two browsers to show the force directed diagram (firefox or chrome) in the browser dropdown.
3. Click on “Generate force directed diagram” button.

GENERATE CLASS DIAGRAM

1. First you need to generate the textual representation.
1. You can choose a profile in the profile dropdown (profile change the amount of element that form a class and a property). There is also a possibility to change it manually with a number from 0 to 100 where 0 mean all elements will be classes and 100 only important element will be classes. 0 properties means that all remaining elements will be shown as properties while 100 means no elements will be shown as properties.
2. There are six Graphviz properties[1]:
    a. **Overlap**: Determines if and how node overlaps should be removed. Nodes are first enlarged using the sep attribute. If "true" , overlaps are retained. If the value is "scale", overlaps are removed by uniformly scaling in x and y. If the value converts to "false", and it is available, Prism, a proximity graph-based algorithm, is used to remove node overlaps. This can also be invoked explicitly with "overlap=prism". This technique starts with a small scaling up, controlled by the overlap_scaling attribute, which can remove a significant portion of the overlap. The prism option also accepts an optional non-negative integer suffix. This can be used to control the number of attempts made at overlap removal. By default, overlap="prism" is equivalent to overlap="prism1000". Setting overlap="prism0" causes only the scaling phase to be run.
    b. **Splines:** Controls how, and if, edges are represented. If true, edges are drawn as splines routed around nodes; if false, edges are drawn as line segments. If set to none or "", no edges are drawn at all. The values line and spline can be used as synonyms for false and true, respectively. In addition, the value polyline specifies that edges should be drawn as polylines. The value ortho specifies edges should be routed as polylines of axis-aligned segments. The value curved specifies edges should be drawn as curved arcs.
    c. **Sep:** Specifies margin to leave around nodes when removing node overlap. This guarantees a minimal non-zero distance between nodes. If the attribute begins with a plus sign '+', an additive margin is specified. That is, "+w,h" causes the node's bounding box to be increased by w points on the left and right sides, and by h points on the top and

[1] Source : http://www.graphviz.org/doc/info/attrs.html

bottom. Without a plus sign, the node is scaled by 1 + w in the x coordinate and 1 + h in the y coordinate. If only a single number is given, this is used for both dimensions.

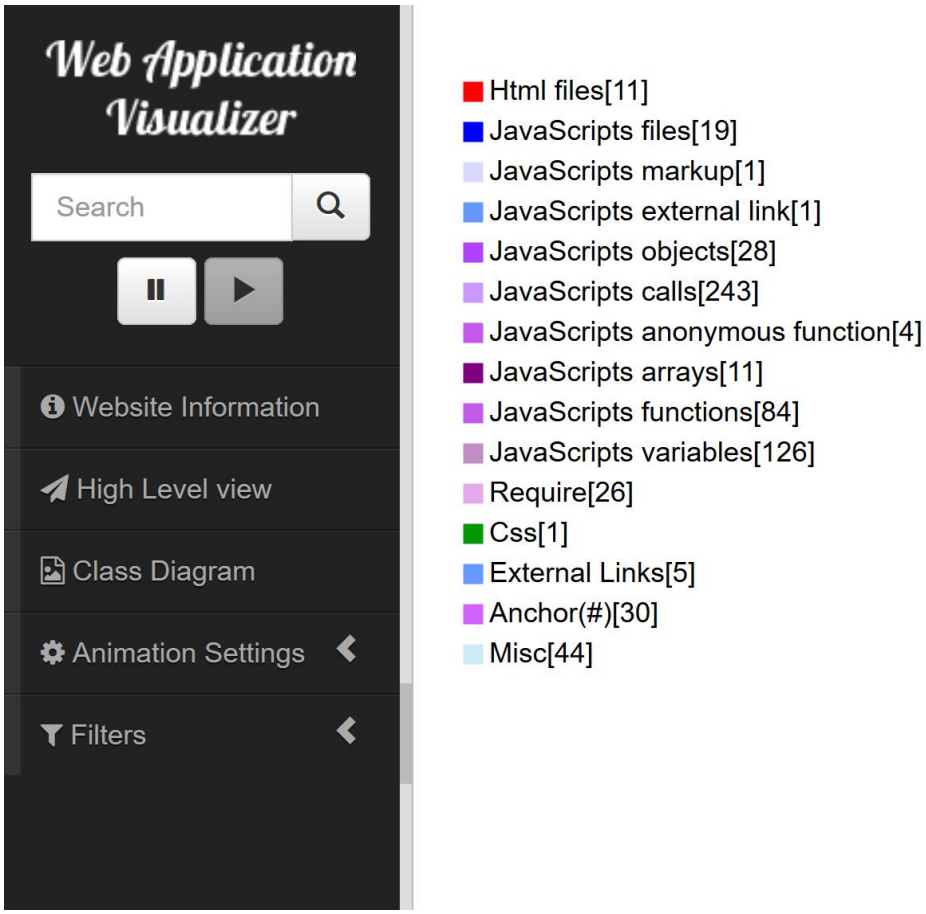
d. **Layout:** Specifies the name of the layout algorithm to use, such as "dot" or "neato". Normally, graphs should be kept independent of a type of layout. In some cases, however, it can be convenient to embed the type of layout desired within the graph. For example, a graph containing position information from a layout might want to record what the associated layout algorithm was.

e. **Dpi :** This specifies the expected number of pixels per inch on a display device. For bitmap output, this guarantees that text rendering will be done more accurately, both in size and in placement. For SVG output, it is used to guarantee that the dimensions in the output correspond to the correct number of points or inches.

f. **Output :** The output is the format of the file “svg”,”png”,”pdf”,”jpeg”. This should be noted that big diagram should be only generated in svg because Graphviz is limited in image dimension.

3. You can check the checkbox « Try to fit in a page » to adjust the graph size to A4 at the best of its ability, sometime it’s just not possible.
4. Click “Generate class diagram”.
5. The diagram is generated in the output folder in the specified format and automatically open with the default editor. If it is in svg and a browser, make sure to scroll left and down because the diagram is centered and if it’s too big, it will look like the web page is empty.

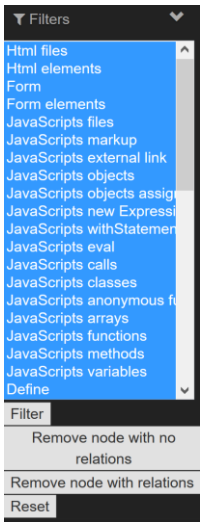
FORCE DIRECTED DIAGRAM



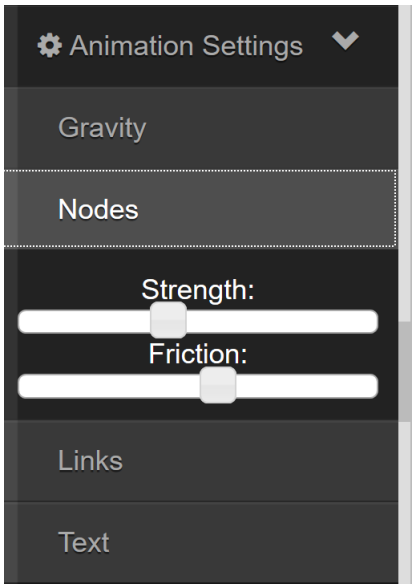
As a first visualization of the structure of a web application, we propose (interactive) force-directed diagrams in which nodes constantly move until equilibrium is reached. They allow a viewer to form a quick opinion about the kind of structure he is looking at, which in some cases can lead to better awareness of potential problems. In force-directed graphs, nodes are all drawn towards the middle but away from each other. The links maintain the connected nodes close to one another. Gradually, as time passes, an algorithm adjusts the position of the elements and the most connected nodes are found in the middle while the least used nodes are found in peripheral.

For WAVI, we generated force-directed diagrams using the library D3.js. Each node represents a source code element and its color represents its type (e.g., HTML files are red, JavaScript files are blue and style sheets are green). To mitigate possible occlusion and performance issues, several options were implemented.

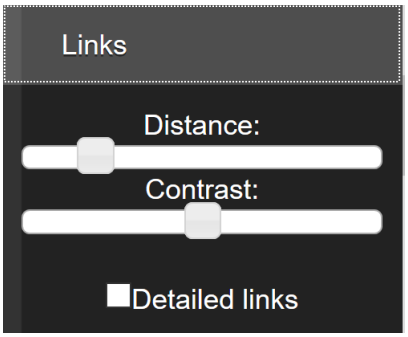
First, there is the possibility to search for a particular file: when you enter a name in the search box, the graph moves to the target node.



Second, it is possible to choose which categories (e.g., class, method, variable) of items are to be displayed.

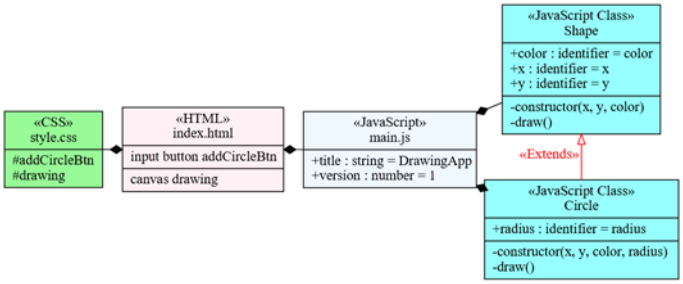


Third, it is possible to modify the parameters of the force-directed graphs, i.e. the force with which the nodes repel each other, the friction controlling the convergence pace or the desired distance between the nodes.



Finally, it is possible to change the contrast of the links (to better see items they are on top of) and change the text size.

CLASS DIAGRAM



WAVI offers a second, more formal visualization: class diagrams customized for web applications as proposed in WAE. The elements are presented in the form of classes and properties. The result is a much more compact diagram (relatively to force-directed diagrams) that contains fewer links, which helps visualize the entire structure of bigger web applications. It is also a more intuitive way to represent the structure of an application because developers are accustomed to seeing class diagrams when programming applications with languages like Java or C ++.

Similar to what we did for force-directed graphs, we provide options for users. In particular, we propose a mechanism to accommodate scalability based on the relative importance of each structure element. To do so, we take into account the type of the element (e.g., class, file), its position in the tree hierarchy, its visibility (public or private), its anonymity (or not), the number of LOC it contains and the number of relationships with other elements. These characteristics are used and combined to assign to each element an importance index between 1 and 100.

The element's type is the most important information. Files and classes have their importance index set to 100, meaning they are always present as nodes while functions and variables may be left out depending on other criteria. The hierarchy level is the second most important information. Here, the hierarchy level of an element is the result of its (containing) file hierarchy (from a directory perspective) combined with its own hierarchy within the file. The more imbricated an element, the less important it is. Visibility and anonymity are useful information as private or anonymous elements are obviously less essential to document. Finally, the number of lines of code and the number of relations are used to weight the importance of an element: for instance, for a function, the greater the LOCs or the number of relationships (e.g. calls), the greater the importance of an element.