**The Open University of Israel**

**Department of Mathematics and Computer Science**

**JXON-D**

JSON and XML Designer for Engineers

A description of an Advanced Project course 22997 submitted as partial fulfillment of the requirements

Towards an M.Sc. degree in Computer Science

The Open University of Israel

Computer Science Division

By

**Gary Draishpits**

Prepared under the supervision of Dr. Leonid Barenboim

2017

Table of Contents

[2 Background 3](#_Toc495480234)

[3 Project Description 4](#_Toc495480235)

[3.1 Purpose 4](#_Toc495480236)

[3.2 Initial Requirements and Guidelines 4](#_Toc495480237)

[3.3 Architecture 4](#_Toc495480238)

[3.3.1 WEB 4](#_Toc495480239)

[3.3.2 Angular 5](#_Toc495480240)

[3.3.3 Client-Server 5](#_Toc495480241)

[3.4 Design and Implementation 5](#_Toc495480242)

[3.4.1 The GUI 5](#_Toc495480243)

[3.4.2 Data Structures 8](#_Toc495480244)

[3.4.3 Main Flow 10](#_Toc495480245)

[3.4.4 Services 11](#_Toc495480246)

[3.4.5 Hiding vs Not Showing 12](#_Toc495480247)

[3.4.6 Components 12](#_Toc495480248)

[3.5 External Libraries 13](#_Toc495480249)

[3.5.1 Drag&Drop 13](#_Toc495480250)

[3.5.2 XML 13](#_Toc495480251)

[3.5.3 Saving a file locally 13](#_Toc495480252)

[4 References 13](#_Toc495480253)

# Background

Text-markup [1] and data-interchange formats are very common, useful and widely used file formats. They have been around for many years and utilized by engineers for purposes such as data classification, storage and standardization.

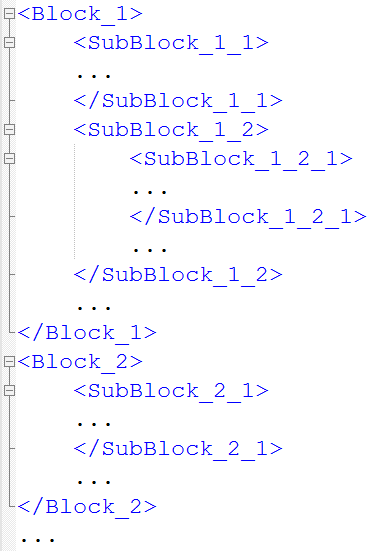
They are particularly useful as a middle-tier between SW layers [2]. One can think about them as the API between the different SW layers. When a SW layer is undergoing a maintenance or even missing, engineers often use the data-interchange files as a temporary mediator. For example, if there is a system that has no graphical user interface, through which users pass commands to it, markup files are a very good temporary solution as they provide a level of standardization with all its benefits. Another example is a missing SW module that handles the Database layer. In this case, engineers may use markup files to store information, and once the DB module is implemented, these standard format files are easy to manipulate in order to move the data to a proper DB.

Figure 1- Markup files as mediators between SW layers

Although these workarounds provide practical solutions, they significantly limit the scalability of a system. A proper GUI is much more resistant to errors than text files manually edited by people. In many cases, engineers are forced to work directly with text files, because production cannot stop and wait for proper SW modules. There are many solutions over the Internet that offer some GUI for manipulating markup files [7][8]. Nevertheless, they often don’t offer a solution that is significantly better than a good text editor [9], while in other cases they provide an overcomplicated system that requires too much investment of resources and time. Eventually, engineers return to the text editor.

# Project Description

## Purpose

The main purpose of this project is to provide an enhanced Graphical User Interface to handle text-markup files that will allow engineers to edit such files in a fast and intuitive manner, while limiting the chances of an error. The solution shall support the two popular file extensions – XML and JSON.

## Initial Requirements and Guidelines

The final solution shall provide the following abilities –

* Collapse and expand the files by their internal blocks structure.
* Change the location of any block, using drag and drop (D&D).

Figure 2- XML Blocks structure

* Move blocks from one file to another, using D&D.
* Support XML and JSON formats, so the users will be practically given a way to change file format from XML to JSON and vice versa.
* Edit the parameters of the file’s blocks.

In order to provide a solution that will present a real value to engineers, the UI shall be very simple, intuitive and elegant, without overloaded button panels and unnecessary features. It shall require extremely minimal, if any, training. It shall be fast and responsive. It shall be modular and portable in a way that will allow engineers to download it to their intranets and connect to their Databases.



Figure - Basic Flow

## Architecture

### WEB

One of the main goals of the solution was that it should be practical. That means that we should reduce the number of dependencies as much as possible. For example – frameworks such as .NET and JRM are very popular yet still limited and vulnerable to Operating Systems and self-versions that may cause an application to stop working at some point of time. An application designed using web technologies may be executed by a standard and most popular environment – a web browser, such as Google Chrome and Mozilla Firefox. It will be OS-agnostic and will be limited only by the abilities of the browsers to execute standard JavaScript. In SW development world there is always a chance for an application to brake due to changes in the frameworks that execute it, yet web technologies are usually more resilient.

### Angular

JXON-D was designed and implemented using the Angular [11] framework and specifically, Angular 2+, as Angular 1 was redesigned and rewritten starting Angular 2. The programming language is Typescript, which is a superset of JavaScript, meaning that it is fully backwards compatible with JavaScript while introducing new capabilities, such as types and object-oriented programming.

Another important offering of the Angular framework is its clear separation of the user interface from the logic and the data. While traditional web design separates the data from the UI successfully by using Databases, it fails to do so with the separation of the presentation layer from the logic layer, often mixing the HTML (UI) with JavaScript (logic) and server side framework, such as PHP.

The eventual solution is an application that is compiled by Angular to standard JavaScript and being executed by a standard browser.

### Client-Server

An important aspect of any web application is the involvement of a server-side logic. One of the possible implementations for JXON-D could be uploading the files to a server and performing any logic requested by the user on the server side. We chose not to implement a server side and rely on client side logic only. This provides the following advantage – users can use JXON-D on their personal computers locally, without bringing-up an HTTP server.

Despite of the fact that there is no server-side logic, since it is a standard web-based project based on JavaScript, the full solution may be placed on a server and opened by the same standard browsers from remote.

## Design and Implementation

### The GUI

As intended, the user interface of JXON-D is very simple, minimalistic and elegant, to allow fast and productive work for the users.

There are two sides – Left and Right. The basic flow is the following –

1. C:\Users\gdryschp\Desktop\Development\Angular4\Gary\jxon-d-TFactory\src\app\images\open-file-white.pngThe User opens a JSON or XML file on the left side.
2. The User opens a JSON or XML file on the right side.
3. Using Drag and Drop, the user moves or copies elements from one side to another.
4. The user downloads a copy of the files to the local storage.



Figure - JXON-D UI - Basic

C:\Users\gdryschp\Desktop\Development\Angular4\Gary\jxon-d-TFactory\src\app\images\new-document-white.pngExcept for editing files by exchanging elements between the files, there is an option to create a new empty file that may be filled by elements from other files.

The number of files that may be opened is not limited.

The UI also provides an option to add new elements that may be used by the users.

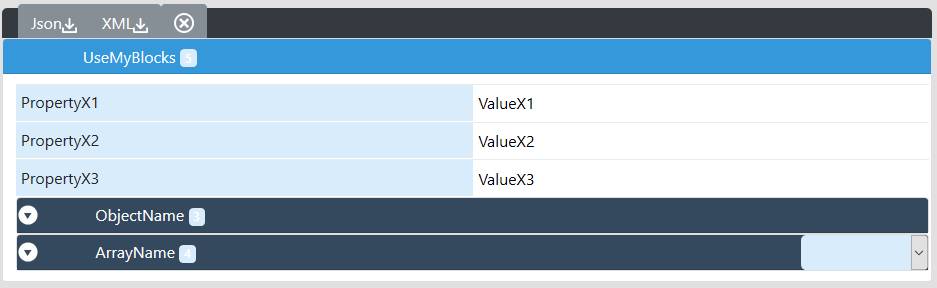


Figure - Elements for Reuse

#### Elements

The UI displays the following elements –

* Property – this element doesn’t have nested elements
* Object – may have nested elements of any kind
* Array – a list of nested elements of any kind

#### Working with Arrays

Array’s nested elements are named simply by a numeric index. This makes very difficult to work with them as the users don’t know what the nested elements are. For that matter we’ve implemented a combo-box for an Array element that exposes the nested values without opening the objects.

When reading a file, Array’s elements are presented as objects. It is possible though to D&D any type of element to the Array, including properties. When the file is saved, the keys of the elements inside the Array GUI element are ignored, and instead, indexes are assigned as appropriate for Arrays.

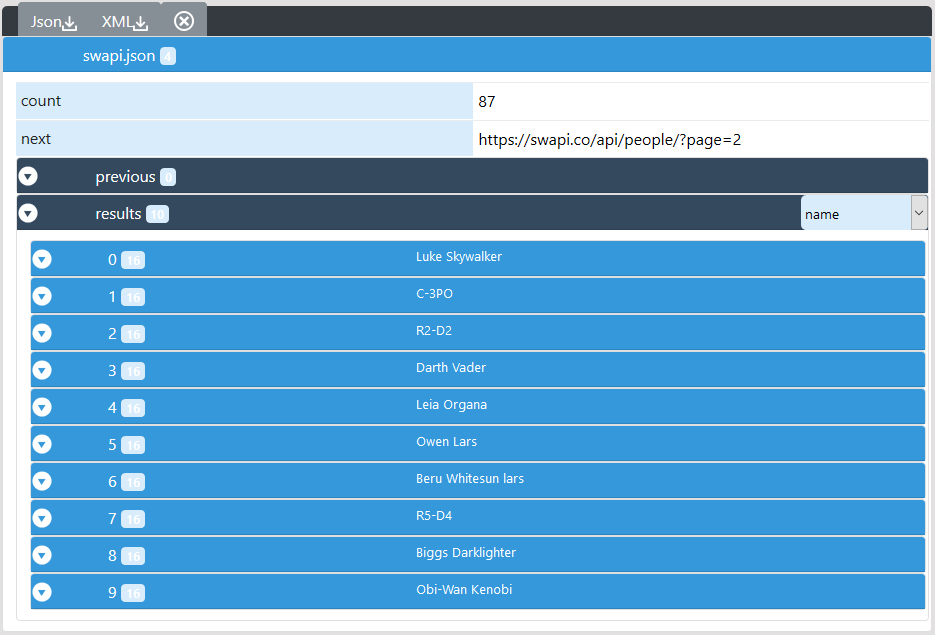
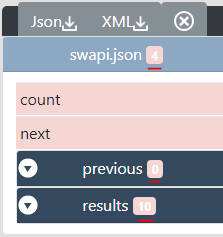


Figure – Array nested elements exposed

#### Nested Elements Count



Near every name of an element of type object or array, there is a small badge indicating how many nested elements are inside. It represents the number of the first level nested elements of all types – objects, arrays and properties.

#### Element Menu

C:\Users\gdryschp\Desktop\Development\Angular4\Gary\jxon-d-TFactory\src\app\images\menu-icon-triangle.png

Each Object or Array element has a menu with the options to Clone, Delete or change its color. The coloring is useful for marking an element during editing.

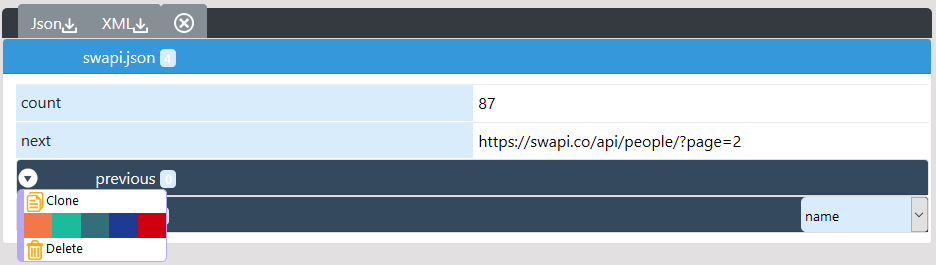


Figure - Element Menu

When an element is cloned, it is placed before the original element, so the user will not have to scroll down in case the element was opened during the cloning. Naturally, an element is cloned with all its internal structure.

#### Drag and Drop (D&D)

For the purpose of changing the structure of the files, JXON-D supports D&D.

Any element can be dragged and dropped within another file, or an element of another file. For obvious reason, elements can’t be dropped inside a property, but in elements of type Object or Array.

As we mentioned previously, when dropping an element in an Array, the element will retain its name. The name will be ignored and changed to an index during the export to an actual file.

Using the toggle button of the top menu, users can specify whether to copy elements during D&D or move them. The second toggle button allows to deny dropping elements on the left side. This is for the use case when there is a source model on the left and users use it to create files on the right, using this model.

Elements may also be reordered using D&D.

### Data Structures

#### Block and BlockComponent

A Component is an Angular term for a UI component. Every element of the user interface is represented by a BlockComponent instance that holds the UI settings of the element, and a Block instance that holds the data itself and the logic to handle the data.

Block instances are created by JXON-D, while BlockComponent instances are created by the Angular framework using specific Angular directives.

Every Block holds the following data –

|  |  |
| --- | --- |
| Parameter | Description |
| blockIndex | The Index of the Block |
| key | Original Key of the Block |
| value | Original Value of the Block |
| fileIndex | The index of the file the Block resides within |
| parentBlock | A reference to the parent Block |
| nestedBlocks | An array of Blocks that represents the nested elements of the element represented by the Block |
| isValueObject | A flag |
| isValueArray | A flag |

#### JxonFile and FileComponent

Every File opened by the user, as well as the New and Reuse Elements offered by JXON-D, are represented by a FileComponent Component that represents the UI part and a JxonFile that represents the data and the logic.

A file holds an array of Blocks.

#### blockIndexes Hash Table

Every Block has an index that is assigned when the Block is instantiated. To increase performance, for example during data structures update following an element drop, we save the pairs of (Block-Index, Block-Instance) in the blockIndexes hash table. Block-Instances are saved by reference, thus this data structure has small effect on the RAM, yet significant impact on performance during a search of a Block.

#### LeftAreaFiles and RightAreaFiles

Two Arrays that hold the opened files. These structures are used to inform the Angular framework of the need to instantiate the FileComponents.

#### filesArea

A Hash Table of the following pairs – fileIndex🡪Area. The Area is either left or right.

### Main Flow



The first level of the nested blocks of a file contains one block only. We still keep it in an array as the user may drag there another block.

### Services

The Angular framework instantiates and destroys UI Components whenever it decides. The data structures that are being used by Jxon-D are kept in Services which are Angular classes which instances are created only once and remain alive throughout the program lifecycle.

#### FileManager

The FileManagerService keeps the LeftAreaFiles and RightAreaFiles which in turn keep the files as instances of JxonFile. Every instance of JxonFile has a list of Blocks.

Besides the data structures, the FileManagerService is responsible for handling the update of the data structures following D&D.



Figure - Right/LeftAreaFiles hold JxonFiles that hold Blocks

#### IndexingService

Responsible for providing the next file index and keeping the blockIndexes hash table.

#### ColorsService

Responsible for the color themes. Fires an event once a color theme is changed by the user.

BlockComponents are registered to the color change event.

#### AlertingService

Responsible for displaying Alerts to the user.

### Hiding vs Not Showing

There were two possible ways to display the blocks of a file. One is to display all blocks in the DOM [14] and hide them using the ‘hidden’ HTML attribute. The advantage of this approach is that once all elements are loaded into the DOM, opening and closing the blocks is fast. The main disadvantage is that opening large or even medium size files, that contain hundreds or thousands of elements, becomes an impossible task and the browser becomes not responsive.

Due to this reason, we chose to implement a second approach, using the Angular ‘\*ngIf’ directive, that instructs Angular to create an element only if it results to true. Using this directive the only elements that exist in the DOM are those that the user actually sees. Any nested element doesn’t even exist, and created only when the user clicks its parent. When the block is clicked and thus opened the nested elements are created and added to the DOM by Angular. Using this approach we were able to open very large files (~100MB) with thousands of elements. The disadvantage of this approach is that nested elements are destroyed and created by Angular every time their parent element is closed or opened.

Another advantage of the second approach is the clear separation of the Presentation and the Logic layers, as now, data structures can’t be stored in a UI Component as it will be destroyed and instantiated per need by the Angular framework. This, in turn, means that any properties will be instantiated to their initial values and any user change will be lost. As previously mentioned, the data structures are kept in services.

### Components

As previously mentioned a Component is an Angular framework element that is being used to present the graphical user interface.

These are the components that are implemented in Jxon-D.

|  |  |
| --- | --- |
| Component | Description |
| BlockComponent | Displays a single Block with a Block-Header and Block-Body that displays all the nested elements when it’s open.  Every object or array are presented by the BlockComponent. |
| FileComponent | Displays a File |
| BlockMenuComponent | The Menu on every BlockComponent |
| HeaderComponent | The main header of the application with the title and the colors themes |
| LoadFileComponent | Displays the buttons panel. There are 2 instances of this component, for every side – Right and Left |
| AppComponent | The main component of an Angular application |

## External Libraries

### Drag&Drop

For drag and drop we chose to use the **Dragula** [11] library that was somewhat popular. Although it is good enough for the basic use case of moving an element from one container to another, it has many issues when using complex elements, such as we use in Jxon-D.

Due to these issues, we were forced to implement several additions for handling copying of an element instead of moving, cloning of an element and delete.

### XML

Since JSON is JavaScript Object Notation, it is natively supported by JavaScript and thus, by Angular. Not so with XML. For handling XML, we’ve used the **xml2js** [12] package for transferring an XML string to a JS object and vice versa.

### Saving a file locally

To save a file locally we’ve used the FileSaver [13] utility package.

# References

1. <https://en.wikipedia.org/wiki/Markup_language>
2. <https://en.wikipedia.org/wiki/Multitier_architecture>
3. <https://www.w3schools.com/xml/xml_whatis.asp>
4. <http://www.json.org/>
5. <https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats>
6. <https://en.wikipedia.org/wiki/Software_prototyping>
7. <http://www.lexiconista.com/xonomy/web-based-xml-editors/>
8. <http://www.jsoneditoronline.org/>
9. <https://notepad-plus-plus.org/>
10. <https://angular.io/>
11. <https://github.com/valor-software/ng2-dragula>
12. <https://www.npmjs.com/package/xml2js>
13. <https://github.com/eligrey/FileSaver.js/>
14. <https://www.w3schools.com/js/js_htmldom.asp>