DOI: 10.15439/2016F574
ACSIS, Vol. 9. ISSN 2300-5963

Medical reporting in web-based applications designed to meet regulatory and industry standards

Michał Madera
Rzeszów University of Technology
al. Powstańców Warszawy 12,
35-959 Rzeszów
Poland
Email: michalmadera@gmail.com

Rafał Tomoń
SoftSystem Sp. z o.o.
ul. Leszka Czarnego 6a,
35-615 Rzeszów
Poland
Email: rtomon@softsystem.pl

Piotr Lorenc
SoftSystem Sp. z o.o.
ul. Leszka Czarnego 6a,
35-615 Rzeszów
Poland
Email: plorenc@softsystem.pl

***Abstract*—Web based applications penetrate into every software domain. Even those reserved only to desktop programs are becoming available through web browsers now. This has brought real technical challenges to software developers. Medical programs are not different. In this paper we are proposing new approach to text processing for web browser based medical applications. We are focusing on entering text of medical interpretation which is very important and sensitive aspect of medical report creation. A number of products were reviewed to justify the need for research in this area. The developed approach integrates report assembling, presentation and diagnosis text processing in accordance with medical data safety regulations. We prove that proposed solution based on HTML5 Canvas can be applied to development of the most demanding pathology reporting applications.**

I. Introduction

DEVELOPMENT of web based applications for reporting in medicine brings variety of challenges. The need for online access to software is unquestionable today. Initiated by the American Recovery and Reinvestment Act (ARRA) by 2009 pertained to areas like patient electronic health record [1] or management reporting. In spite of problems [2] online access to medical software tends to cover all areas nowadays. We will focus on challenges with porting medical reporting systems to web based applications. For the use of this paper we generalize medical report document structure [3]. The reports we are considering here can be divided into the following parts:

1. The Patient Information part containing patient demographics and clinical information (medical record number, attending doctor, etc.),
2. The Diagnostic Tests part which refers to laboratory testing results (including historical results) and observations,
3. The Medical Diagnosis part which is medical interpretation text entered by health care professionals (pathologist, radiologist, medical laboratory scientist, etc.) that we will refer to as diagnostician,
4. Report Header and Footer sections that usually contains medical institution information.

This work was supported by SoftSystem Sp. z o.o.

General template for medical report and its parts is presented in Fig. 1.

There is a wide context of medical reporting and problems we faced working with such systems. In this paper we

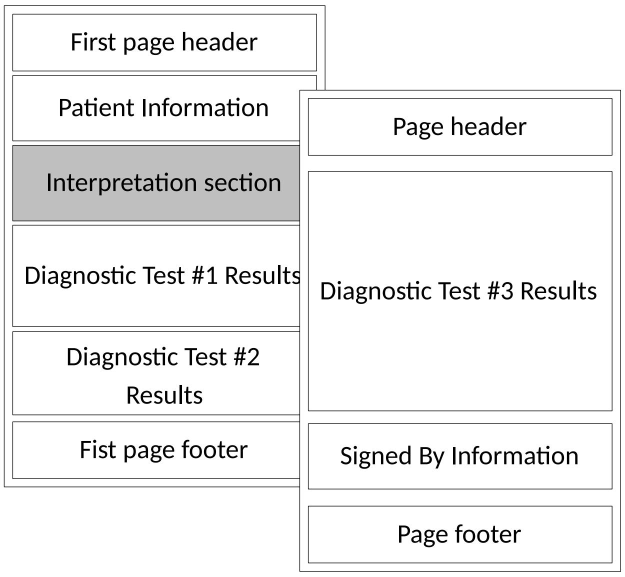


Fig 1. Medical report document structure

decided to focus on issues which are causing difficulties, especially in the web based environment.

Among the issues selected by Paul N. Valenstain, MD [4], the following are significant ones for which a solution will be presented in this paper:

1. When diagnostic tests are completed and report content is presented to a diagnostician, only the Interpretation sections can be modified (Interpretation can consist of multiple sections). All other sections of the document can not be modified (Diagnostic Tests, Patient Demographics and Report Header/Footer). Changing diagnostic or patient in-

formation would be considered as safety breach. We will refer to a problem of editable and not editable sections.

2. Complete Report layout has to be visible during interpretation entering and not changed when report is being printed or exported to PDF document. This is the concept of WYSIWYG (What You See Is What You Get).
3. Document presentation should not change across different web browsers, which is one of the most difficult issues to solve. Covering all notable web browsers and different release versions is one of the main requirement. Following browsers are taken into account: Chrome, FireFox, Opera, Safari, Internet Explorer and Microsoft Edge. Evaluation of components available on the market proved that research in this area is justified. Among others, following products were evaluated: DevExpress Html Editor [5], TxTextControl [6], TinyMCE [7]. All solutions based on ActiveX and Sliverlight technologies were not taken into consideration as these technologies are to be withdrawn.

II. Solution overview

The core of the research was to create software component providing the ability to view and edit medical reports in web browsers. It have to allow viewing the document in its final shape and editing in the same time, assuring the requirements listed in this paper's introduction are met.

Achievement was to develop web browser component based on HTML5 Canvas technology for text processing. Using Canvas for text processing is not a common approach with many obstacles that had to be overcome. We investigated possibility of using Canvas custom text drawing component in connection with innovative report merging mechanism. Medical data consistency and safety is guaranteed by presentation of not editable parts of the report as immutable images in conjunction with editable parts completely controlled by our canvas text processing editor. Immutable medical data images provides safety across all types of devices, while commonly used in web environment HTML rendering depends strongly on web browser engine and operating system. That may cause the content to be displayed differently on different devices, as depicted in chapter VI of this paper. For most of the content presented in web browsers this is not an issue. Unpredictable change to the formatting of medical data can lead to incorrect interpretation of the results. Using the same report merging mechanism to generate final PDF document and parts of the not editable report is the biggest gain of the presented solution. Reliability of presented information was the main goal and it was achieved.

III. Device independent document rendering

We defined the following components in our solution:

- *Reporting Engine* is a high-level text processing and reporting component for server-based application;
- *Web report* – final document divided into logical slices like header/footers, non-editable and editable slices;
- Shark Editor – text editor developed based on open source project named Carota.[8];
- *Shark JSON* – internal text editor format based on open-standard data format JSON defined by RFC 7159 [9];
- *Canvas* – element of HTML5 specification that allows dynamic, scriptable rendering of 2D shapes and bitmap images [10];
- *Font maps* – a set of properties described fonts like Times New Roman, Arial, Verdana, etc.

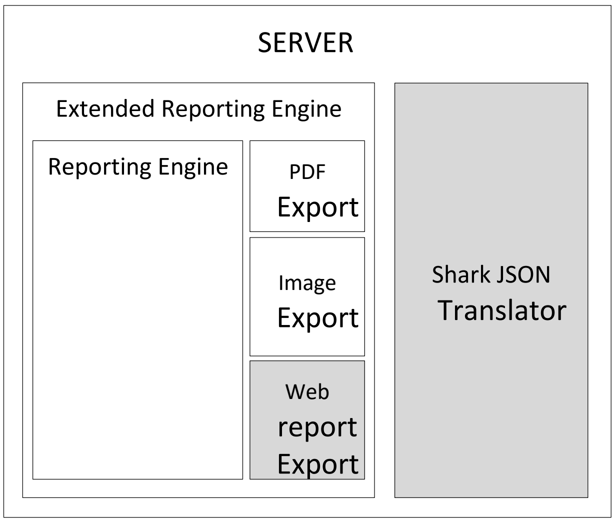


Fig 2. Server architecture

The architecture of the medical reporting system is a typical client-server architecture. System generates report on the server and returns it to the client using TCP/IP protocol. The client application, located on a web browser, presents the report to the user and handles all user inputs. Fig. 2 and Fig. 3 show graphical representation of the system architecture. The main element of server side application is a reporting engine which can provide fully merged medical report. The reporting engine was extended to provide the ability to export document in a new format – web report format. The advantage of this solution is fact that report engine is replaceable part of architecture. System may use any reporting engine available on market. Two requirements for the engine are: 1) ability to export document as image; 2) discreet representation of document in memory;

The client application was developed using HTML5 technology. Web report is passed to client in a JSON format, which is the most common data format used in browser/server communication. Text Editor Engine creates a virtual document with words, lines and sections, then measures it using provided font map, and then renders the document on the screen. Each element (word, letter, image, etc.) is drawn directly on the canvas that gives absolute control over what is presented on the screen and how it reacts to user changes.

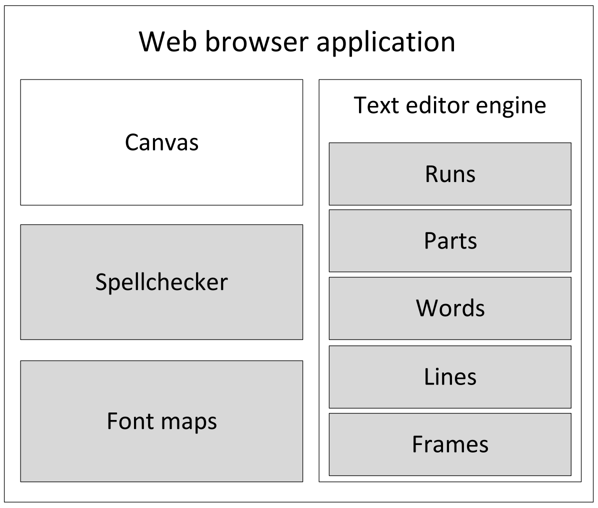


Fig 3. Web Client architecture

IV. Exporting the document

The web report is generated on server using the extended reporting engine. When the document is fully merged, the system is exporting it into bitmap image. Then using discreet data it is searching for logical slices: headers/footers, non-editable document content (diagnostic tests) and editable medical diagnoses.

The main attribute of logical slice is area boundary. This is a value calculated from the discreet representation of data. This attribute is used by server component to cut the image document exported earlier. Some of slices may occur on every page (like headers or footers) and to improve performance of exporting, the system cuts it only once. Cut slices are sorted in the order they occurred on the merged document. The generated web report contains information about whole document but it is divided into logical slices sorted chronologically.

Non-editable slices like headers/footers and diagnostic tests are exported as image fragments, but editable medical diagnoses are exported as RTF (Rich Text Format). System uses RTF, because RTF is a common format for text representation in medical software [11]. Diagnostic tests sections exported as image guarantee that the laboratory testing results will be modified neither by end user nor by malicious software.

The web report format may be used in text editors developed in other technologies like (X)HTML, WPF, Swing, WinForms. In our approach we use web report format to display it on canvas item and translate it to JSON format.

V. Canvas and document structure

Web Report in JSON format consists primarily of an array of JSON objects, called runs. They are parts of text with consistent formatting as well as some special elements (markups, images, etc). This format supports most commonly used properties, like font sizes and styles, colors and alignment. A sample text: “Plasma glucose is **about 12% greater** than...” is presented in JSON as follows:

```
[
   {   "text": "Plasma glucose is " },
   {   "text": "about 12% greater",
       "bold": true },
   {   "text": " than..." }
]
```

Elements in curly brackets are called *runs* and text from the sample is represented by three *runs*. Besides such simple objects, document structure also supports two complex types of *runs*. The first of them is also text based, but provides custom behavior on user input, e.g. could be modified only using drop-down or editor pop-up. It can be used to merge and maintain always up to date patient details, or provide some predefined values. This element contains extra attributes like *type* and *fielddata*. Below is an example of such item in JSON format.

```
{  "text": " Patient Last Name",
   "type": "TEXTFIELD",
   "fielddata": {
      "type": "MERGEDFIELD",
      "source": "PatientLastName" } }
```

The second type of *run* is not printed directly on the screen, but provides logical division of the document and ability to assign custom behavior. Used mostly to define read-only/editable items or distinct different parts of the report:

```
{  "text": {  "$": "sectionStart" },
   "code": "_SEC_GROSS_DESC" },
{  "text": "Interpretation text" },
{  "text": {  "$": "sectionEnd" } }
```

While the document loads, collection of runs is transformed into virtual structure of characters, words and lines which are stored in memory, to improve rendering performance. Each of these items have measured their own boundaries using a provided font map with ascent/descent values as well as width/height dimensions (Fig. 4). They are determined by the size of nested objects, so height of a word comes from the height of its letters, and the width is a sum of letters widths. Line dimensions are determined by the nested words, documents by nested lines.

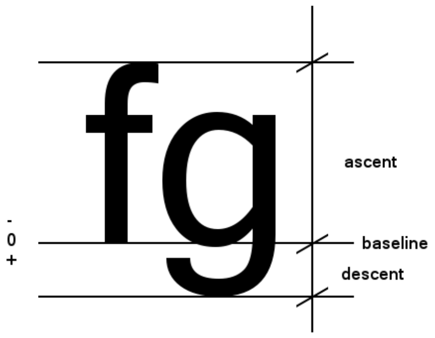


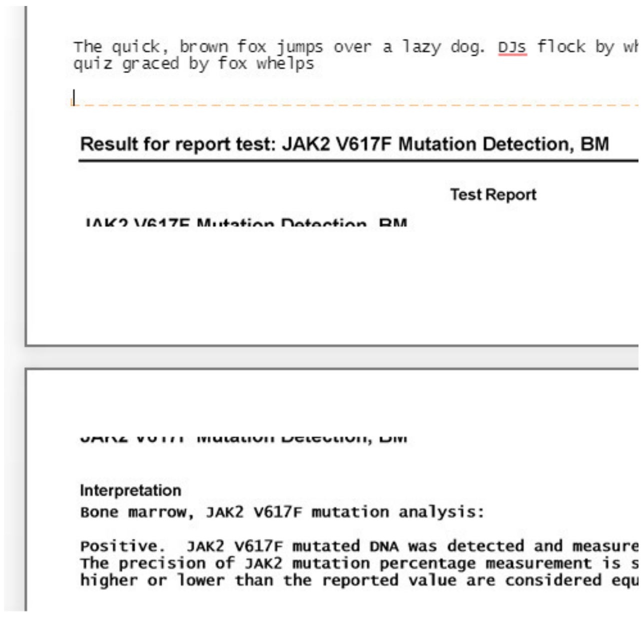
Fig 4. Font map to measure text size

Using font map not only speed up building of document structure, but also improve matching between canvas document and final PDF report.

JSON data also contains information about document layout, like paper size and margins, used to split content into words, lines and pages in the same manner, as in Reporting Engine. When the document structure is ready, visible part of the report (based on scroll position and window size) is drawn on the canvas, using its native methods like *fillText* or *drawImage* with measured sizes. Whole process starts from the document objects, that initiate drawing its lines and lines draws its words, etc. When the user is typing, entered text is first handled by hidden HTML text area component, which triggers document update – current word is immediately measured and the whole structure is updated.

Wrapping content and text between pages is divided into two steps, to maintain the best fit with final report. First, content is wrapped dynamically based only on current editor state (page size, line height, left space). This is fast, but can sometimes lead to unexpected results, such as splitting of non-editable, but consistent content, that should be moved as a whole to the next page. Fig. 5 demonstrate this situation.

When the user stops typing for a moment, a synchronization mechanism is triggered, as second step of this process. It requests the current pages split points from the PDF engine, compares them with current state and perform some minor additional adjustments, if needed. Fig. 6 demonstrate the result document after the synchronization. This mechanism guarantees that one of the postulates is met – "What You See Is What You Get".


The quick, brown fox jumps over a lazy dog. DJs flock by wh
quiz graced by fox whelps

Result for report test: JAK2 V617F Mutation Detection, BM

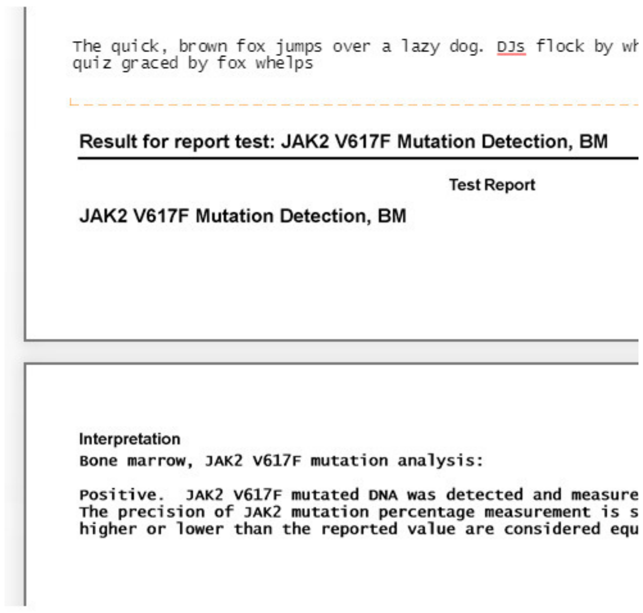
Test Report

JAK2 V617F Mutation Detection, BM

Interpretation
Bone marrow, JAK2 V617F mutation analysis:

Positive. JAK2 V617F mutated DNA was detected and measure
The precision of JAK2 mutation percentage measurement is s
higher or lower than the reported value are considered equ

Fig 5. Page split while typing


The quick, brown fox jumps over a lazy dog. DJs flock by wh
quiz graced by fox whelps

Result for report test: JAK2 V617F Mutation Detection, BM

Test Report

JAK2 V617F Mutation Detection, BM

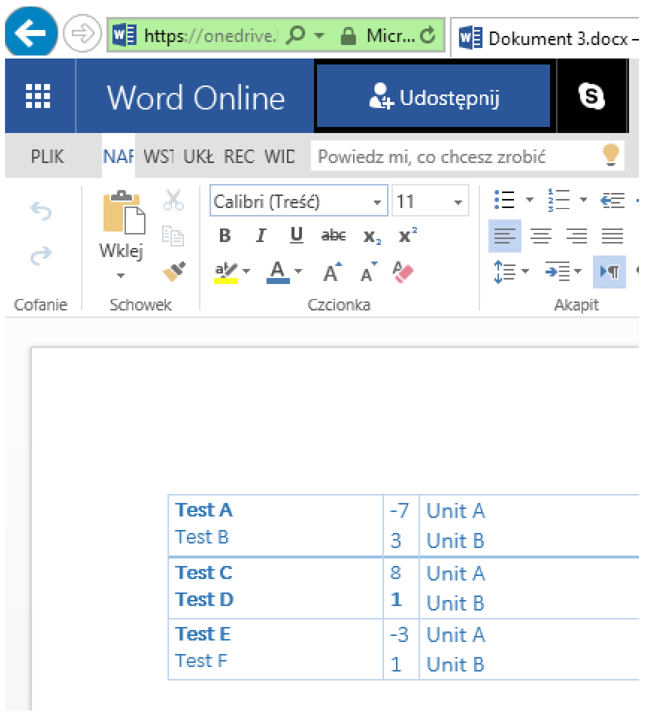
Interpretation
Bone marrow, JAK2 V617F mutation analysis:

Positive. JAK2 V617F mutated DNA was detected and measure
The precision of JAK2 mutation percentage measurement is s
higher or lower than the reported value are considered equ

Fig 6. Page split resynchronized

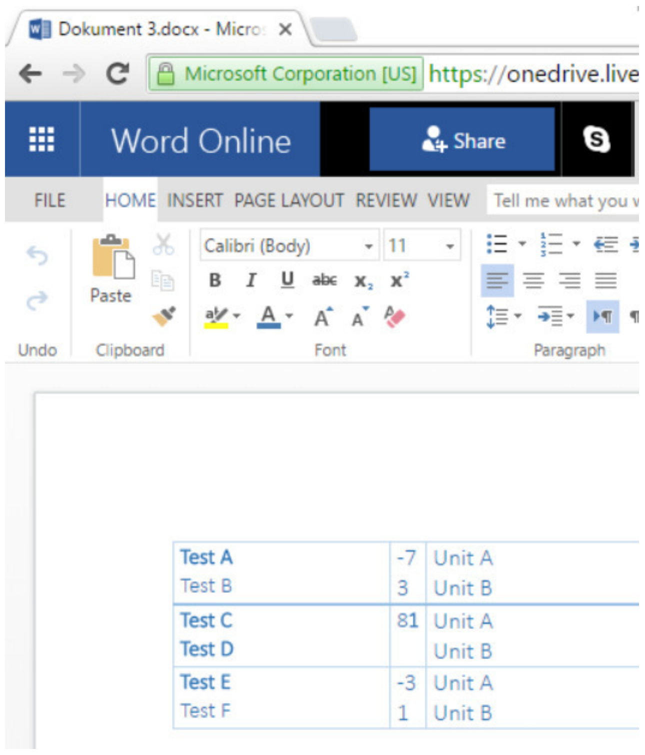
VI. Evaluation

Evaluation of other components proved our solution to be superior. Taking leading product, Microsoft Word Online as an example, we prepared document with tabular data and opened in Internet Explorer browser Fig. 7. Difference in display when opened in Google Chrome is clearly visible in Fig. 8. Presented example demonstrates very dangerous change is data presentation. Due to different interpretation of HTML table size by web browser engine, numbers were consolidated in one line causing error prone situation. As presented in Fig. 9 the same document exported to PDF can change even more.



Test A	-7	Unit A
Test B	3	Unit B
Test C	8	Unit A
Test D	1	Unit B
Test E	-3	Unit A
Test F	1	Unit B

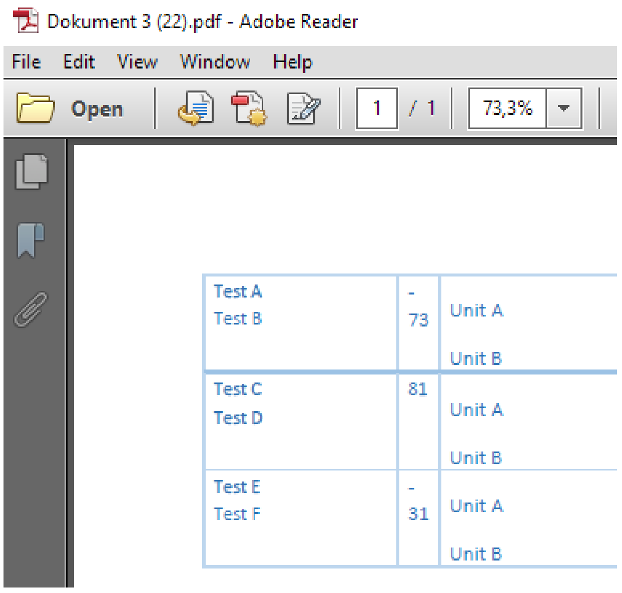
Fig 7. Document presented in Internet Explorer browser



Test A	-7	Unit A
Test B	3	Unit B
Test C	81	Unit A
Test D		Unit B
Test E	-3	Unit A
Test F	1	Unit B

Fig 8. Document presented in Chrome browser

Such behavior is not acceptable for medical reporting. Described example would be classified as Risk to Health incident. Document processing we are presenting in our solution is designed to eliminate similar danger. Presenting of diagnostic data as images makes it platform independent.



Test A Test B	- 73	Unit A Unit B
Test C Test D	81	Unit A Unit B
Test E Test F	- 31	Unit A Unit B

Fig 9. Document presented after exporting to PDF

VII. Results and future work

In this paper, a Report Editor Component for medical web applications was presented. We implemented the prototype and evaluated it in terms of useability. The component is fully functional with a satisfactory user experience. Comparison to Microsoft Word Online [12] and Google Documents [13] was done as these were considered to be a reference implementation of online text processing solutions. When editing large documents the response time has been evaluated as one of the most important aspect of usability. Significant performance improvement was achieved by the novel font mapping mechanism we developed. Support of medical dictionaries is added with "spell as you type" functionality. Initial evaluation of dictation was done but further work is necessary in this area. Still the main advantage of proposed solution is the safety of processed medical information. Design of the component provides extensible base for building sophisticated reporting software for medical industry.

One of the biggest challenges we have to face is table support. Handling tables, nested tables and table cells merging may be one of the most difficult task to complete. At this stage the component we designed can be provided to medical personnel for evaluation and suggestions. Feedback from pathologists would drive further work on this solution.

References

[1] T. Piliouras, A. Fortino, M. Andonov, and H. Huang, “Methodology to assist physicians in the selection of electronic health records software,” in Applications and Technology Conference (LISAT), 2010 Long Island Systems, 2010, pp. 1–6.

[2] S. Ajami and T. Bagheri-Tadi, “Barriers for Adopting Electronic Health Records (EHRs) by Physicians,” Acta Inform. Medica, vol. 21, no. 2, pp. 129–134, 2013.

[3] “Pathology Reports,” National Cancer Institute. [Online]. Available: http://www.cancer.gov/about-cancer/diagnosis-staging/diagnosis/pathology-reports-fact-sheet. [Accessed: 26-May-2016].

[4] P. N. Valenstein, “Formatting pathology reports: applying four design principles to improve communication and patient safety,” Arch. Pathol. Lab. Med., vol. 132, no. 1, pp. 84–94, Jan. 2008.

[5] “Rich-Text and Html WYSIWYG Content Editing - ASP.NET MVC HTML Rich-Text Content Editor Demo | DevExpress.” [Online]. Available: https://demos.devexpress.com/MVCxHTMLEditorDemos/Features/Features. [Accessed: 26-May-2016].

[6] “Text Control - .NET Reporting and Word Processing Components for Developers of Windows, Web and Mobile Applications (TX Text Control) | www.textcontrol.com.” [Online]. Available: http://www.textcontrol.com/en_US/. [Accessed: 26-May-2016].

[7] “TinyMCE | The Most Advanced WYSIWYG HTML Editor.” [Online]. Available: https://www.tinymce.com/. [Accessed: 26-May-2016].

[8] “danielearwicker/carota,” GitHub. [Online]. Available: https://github.com/danielearwicker/carota. [Accessed: 26-May-2016].

[9] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format.” [Online]. Available: https://tools.ietf.org/html/rfc7159. [Accessed: 26-May-2016].

[10] “HTML Canvas 2D Context.” [Online]. Available: https://www.w3.org/TR/2dcontext/. [Accessed: 26-May-2016].

[11] By Laura Bryan CMT BS Technology for the Medical Transcriptionist. Lippincott Williams & Wilkins Publishers, 2009.

[12] “Microsoft Word Online - Work together on Word documents.” [Online]. Available: https://office.live.com/start/Word.aspx. [Accessed: 30-May-2016].

[13] “Dokumenty Google – twórz i edytuj bezpłatnie dokumenty online.” [Online]. Available: https://www.google.pl/intl/pl/docs/about/. [Accessed: 30-May-2016].