Manuscript File Click here to view linked References

XReport: An online structured reporting platform for radiologists

Title / name of your software

XReport

Authors / main developers (incl. affiliations, addresses, email)

[1] Ahmed Harmouche (corresponding author), Department of Radiology, Medical School, University of Pécs; email address: ahmedharmouche92@gmail.com; mailing address: UP MS Department of Radiology: Hungary, 7624 Pécs, Ifjúság str. 13. phone number: +3630/8838435

[2] Ferenc Kövér, Pécs Diagnostic Center; email address: ferenc.kover@gmail.com; mailing address: Pécs Diagnostic Center: Hungary, 7623 Pécs, Rét str. 2. phone number: +3672/242312

[3] Sándor Szukits, Department of Radiology, Medical School, University of Pécs; email address: szukits.sandor@pte.hu; mailing address: UP MS Department of Radiology: Hungary, 7624 Pécs, Ifjúság str. 13. phone number: +3672/536197

[4] Tamás Dóczi, Department of Neurosurgery, Medical School, University of Pécs; email address: doczi.tamas@pte.hu; mailing address: UP MS Department of Neurosurgery: Hungary, 7623 Pécs, Rét str. 2. phone number: +3672/535900

[5] Péter Bogner, Department of Radiology, Medical School, University of Pécs; email address: bogner.peter@pte.hu; mailing address: UP MS Department of Radiology: Hungary, 7624 Pécs, Ifjúság str. 13. phone number: +3672/535801

[6] Arnold Tóth, Department of Radiology, Medical School, University of Pécs; email address: prsarn@gmail.com; mailing address: UP MS Department of Radiology: Hungary, 7624 Pécs, Ifjúság str. 13.

Abstract.

Currently the most widespread way of reporting in radiology is dictation mainly due to performance benefits. The output of this method is plain text, which varies in style (structure, nomenclature, abbreviations, etc.) and content between doctors even when reporting the exact same case. Templated radiology provides a structure for reporting and aims to help in generating more unified reports. We propose a web-based system for creating and using radiological structured reporting templates.

We developed our software based on web technologies. We wrote the system with modular design in mind. We have separate libraries for the different functionalities: a rendering library which renders the templates based on a schema, an editor library which handles template creation, and an evaluator library, which parses, and executes our custom domain specific language, FormScript, which enables dynamic behaviour in our templates. We also developed a Single Page Application to create, browse, use and share templating reports. The backend of the application is powered by Firebase from Google.

We deployed our system at a publicly accessible domain at https://app.radiosheets.com.

Keywords:

structured reporting; radiology; eHealth; JavaScript

Required Metadata

Current code version

Table 1 – Code metadata (mandatory)

Nr	Code metadata description	***Please fill in this column***
C1	Current code version	*v1.4.1*
C2	Permanent link to code/repository used of this code version	https://github.com/wpmed92/xreport
C3	Code Ocean compute capsule	*Not available*
C4	Legal Code License	*MIT License*
C5	Code versioning system used	*git*
C6	Software code languages, tools, and services used	*JavaScript, TypeScript, HTML5, CSS, Bootstrap, Firebase, npm*
C7	Compilation requirements, operating environments & dependencies	*Webpack, Node.js, npm, Angular, Firebase*
C8	If available Link to developer documentation/manual	*https://wpmed92.github.io/xreport/*
C9	Support email for questions	ahmedharmouche92@gmail.com

1. Motivation and Significance

There is no consensus among radiologists on what a good radiological report is. Both radiologists and clinicians who receive the reports have different views on the optimal layout and content. [1] Currently the most preferred way of reporting is dictation. With the advance of speech recognition technology, it became faster, more accurate and easier to produce reports by dictating than by typing. [2] One of the key features of dictation is that it eliminates context switching. Radiologists do not have to take their eyes off the image at any point during reporting, whereas in case of typing they switch between looking at the keyboard and looking at the screen. However, the problem with both dictation and typing is that the output is plain text. Saving large amounts of reports in the Health Information System (HIS) in plain text will generate an archive which is not maintainable, and not searchable. Valuable information will be lost. Also, as mentioned earlier, no two radiologists will write the same report about the exact same image. The use of different nomenclatures, ordering of findings and abbreviations may result in confusion among doctors, and inefficiency in communication and patient management. To address these issues another form of reporting aroused: structured reporting. Structured reporting gives doctors a framework for writing reports. Mostly this framework is template based. A good example of this is the RadReport reporting template collection created by the Radiological Society of North America (RSNA). [3] The collection contains templates grouped by specialties. The templates are submitted by the users and then are reviewed by the Template Library Advisory Panel to ensure the submissions meet certain criteria. The templates are composed of sections, subsections, and input fields. The report is generated by filling the form. Using such templates for reporting have multiple benefits. They can help give radiologists a guideline on what the report should include for a given pathology or modality thus diminishing the possibility of missing some important findings or information. Furthermore, extracting data from such templates and saving it in a database is straightforward, as opposed to plain text reports.

The drawback to templated radiology is that it is hard to find the optimum of how much a report should be structured. If one tried to cover all the possible cases and logical branches using built-in input fields with predefined options, it would be too time consuming to write the report. But if one used mostly text areas, the structured report would resemble a dictation template, and the benefits of structured reporting would be less significant.

We propose a new radiological structured reporting software that is free, cross-platform, can be integrated into the dictation-based reporting workflow of a radiologist, enables template creation and report generation. With our solution we aim to make structured reporting more widespread and accessible, thus increasing the quality and consistency of radiological reports.

2. Software Description

2.1. Software Architecture

The software was written as a web application to support all operating systems and devices. Two programming languages were used throughout the development process, namely JavaScript and TypeScript.

The project can be divided into two main parts: the library and the application. The library is a standalone module that implements the core features of the software: template building and reporting. The application can be any host that integrates the library, in our case it is a Single Page Application (SPA). Our workflow of creating reporting templates resembles intentional programming [4]. The programmer builds the foundation (template builder) of the software on top of which the domain expert (radiologist) can build the actual application (template). The programmer can later add if-else logic to the template.

The library

The library exposes four public methods to interact with: makeWidget, togglePreviewMode, getReportAsText and getTemplateForUpload. The entry point is makeWidget. It can instantiate a new empty template builder or load a template from a Uniform Resource Locator (URL). Internally it creates an instance of each of the following classes: XReportDOM, XReportRender, Evaluator. The XReportDOM implements a custom subset of the Document Object Model (DOM) which allows only specific elements of the DOM or compositions of DOM elements to be used. The XReportRender calls the render methods of the XReportDOM entities and uses them to assemble either a builder or a viewer component, depending on whether the library is in editor or viewer mode. In editor mode the templates can be modified, whereas in viewer mode they are read-only, and are ready to generate reports. The Evaluator is an interpreter for our Domain Specific Language (DSL) called FormScript. It adds dynamic behaviour to the templates through simple if-else logics and calculations. An example of a typical use case for FormScript is to show or hide a specific field if certain conditions are met, or to calculate a score for a scoring system. To view the generated report the library exposes the togglePreviewMode method. Calling this method will transfer the viewer from reporting state to output state or vice versa. In output state the reporter can see the textual output of the form. The generated text can be accessed by the getReportAsText function. When the library is in editor mode a template can be saved by first getting it in JavaScript Object Notation (JSON) format with getTemplateForUpload and then sending it to a web service or storing it locally.

Template structure

The templates are composed of rows, which may have one or more groups in it. Groups are label-entity pairs, and entities are the form's input elements.

The JSON)structure of a template is as following:

{ "formScript": "Form script source code is here", report: [{ XFormElem #1 }, { XFormElem #2 }…]}

General fields in XFormElem:

- type: defines what element to render, e.g. row, group, sel, mulsel
- id: a random generated unique identifier
- scriptAlias: an identifier/variable name by which FormScript can reference the field; auto-generated, but can be changed by user
- hideFromOutput: determines whether the value of the field should be visible in the generated text output
- hidden: determines whether the field should be rendered
- children: a list of groups in a row
- child: the entity of a group

There are fields specific to each entity but they are not listed here.

FormScript

FormScript is a DSL that is specifically designed to run inside XReport templates. It allows custom logic to be executed safely in forms, thus adding dynamic behaviour to them. The script can be edited when the library is in editor mode and is accessible through the getScript library call. Once saved, it is stored in the same JSON file as the template itself.

The FormScript syntax is similar to that of JavaScript with some syntactical differences shown in Table 1.

Features	**JavaScript**	**FormScript**
Logical and	&&	and
Logical or	||	or
Power	**	^
if expression	if (a == b) { … }	if a == b { … }

Table 1 Syntactical comparison of FormScript and JavaScript

Supported binary operations: addition (+), subtraction (-), division (/), multiplication (*), modulo (%), less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equal to (==), logical and (and), logical or (or), to the power of (^)

Unary operations: unary not (!), unary minus (-), unary plus (+)

Statements: expression, assignment, if, function call

Types: string, boolean, number

Numerical and string literals are supported. The only variables that are allowed in FormScript are references to form elements. As mentioned earlier, variables are defined in the editor through the scriptAlias property. Function calls are defined only on variables. It is not allowed to declare functions neither are there predefined library functions without an element context. Calling a function has the following form: variable.function(…parameters). Functions should be defined for XFormElem classes. When XReport loads a report, it checks for an attached script. If there is a script attachment, it will start running it in an Evaluator instance.

Backend

Our SPA has a backend powered by Google Firebase to store the template resources. We store the template files in storage buckets. The metadata for each template, such as date of creation, creator's username, template name, template category is saved to Cloud Firestore documents.

The process of uploading a template to our backend includes the following steps:

- query template JSON from the library through getTemplateForUpload
- assemble upload metadata: date of creation, category, username, template name, template URL
- save the metadata to a Cloud Firestore document
- upload the template JSON file to the storage

Frontend

The frontend is built as a SPA using the Angular [5] and the Bootstrap Cascading Style Sheets (CSS) frameworks. Every icon used in the app are taken from the Font Awesome icon library. Angular supports client-side navigation, asynchronous data binding among others, which enables us to easily fetch and render views. To retrieve templates from Cloud Firestore we use the official Firebase JavaScript Software Development Kit (SDK) and the RxJS reactive programming library. In Firebase terms the templates form a collection, and individual entries in this collection are documents. To show these documents on the screen we followed the Model-View-ViewModel pattern with data binding.

3. Illustrative Examples

Template viewing and building

The viewer/builder page is where we load our templates and render them with our library as show in Figure 1. The form is centered horizontally and have a slight drop shadow around it. There is a button group on the right side of the form which contains different buttons based on which state the page is currently in (viewer or builder).

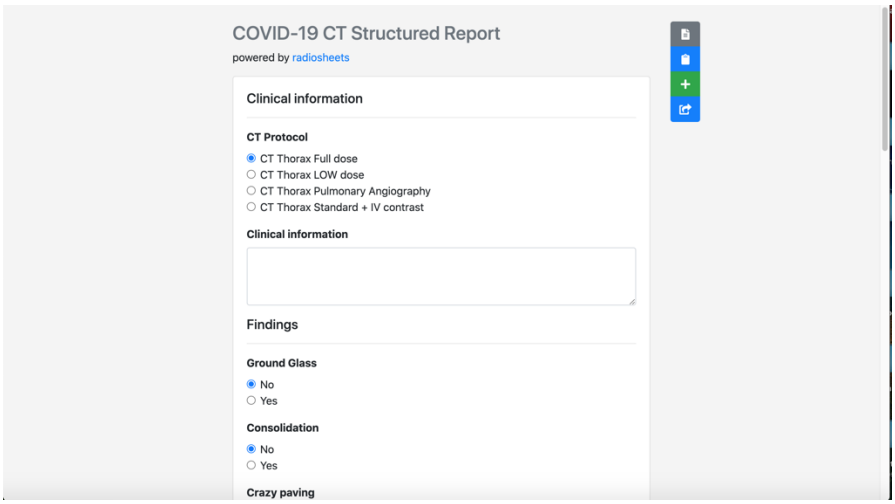

Figure 1 A COVID-19 CT template rendered in XReport

Buttons in viewer state:

- preview report
- copy to clipboard (copies the report output to clipboard)
- new report
- share (copy the template link to clipboard so that it can be shared)

Buttons in builder state:

- save template
- discard template

Builder components

There are 13 components to choose from when building a template:

- Text field
- Plain text
- Number field
- Calculated field
- Boolean field
- Single choice
- Multiple choice
- Textarea
- Date
- Header
- Information
- Rating scale

- Image

If we take an oncological example, a question regarding the size of the tumor would be a number field, a TNM staging system could be created using single choice fields, or a rating scale, etc. Every component is added to the form with a label attached. A component-label combination is called a group. Groups are added to rows, and rows are added to sections. The sections make up the whole template.

Component editors

Every component has an editor view as shown in Figure 2. Every component type has its own editable properties. For example, the input field has a unit property (mm, cm, etc.), a single choice field has an options property, an image has an URL property. The component editor is activated by hovering the mouse over the component, then clicking on the pencil icon. Components can be deleted by clicking on the minus sign.

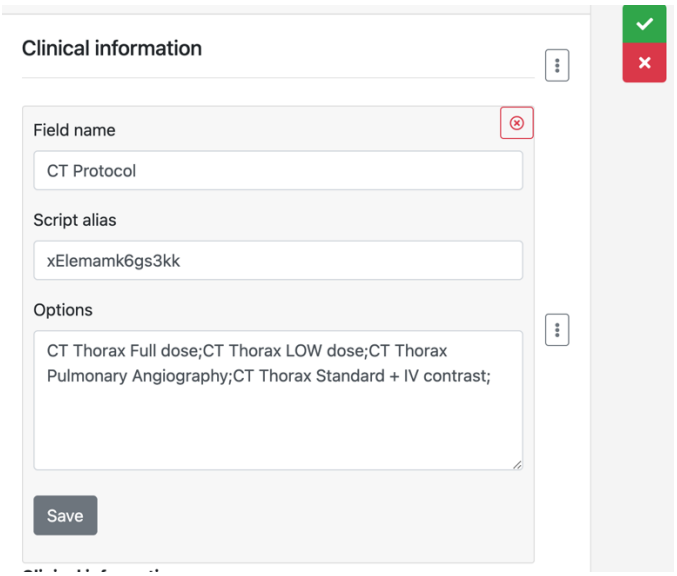

Figure 2 A component editor for a single choice input field

Row editors

Row operations can be performed by clicking on the three vertical dots at the end of each row. The click event will trigger a secondary menu to open with all the components, and two actions: delete and duplicate.

FormScript editor

On the main builder component there is a button with a branch icon which toggles the view between template editing and FormScript editing. The FormScript editor as shown in Figure 3 is a simple resizable text area where the user can edit the dynamic logic that is attached to the template. When switching back from script editing, the script is automatically evaluated and the changes are visible.

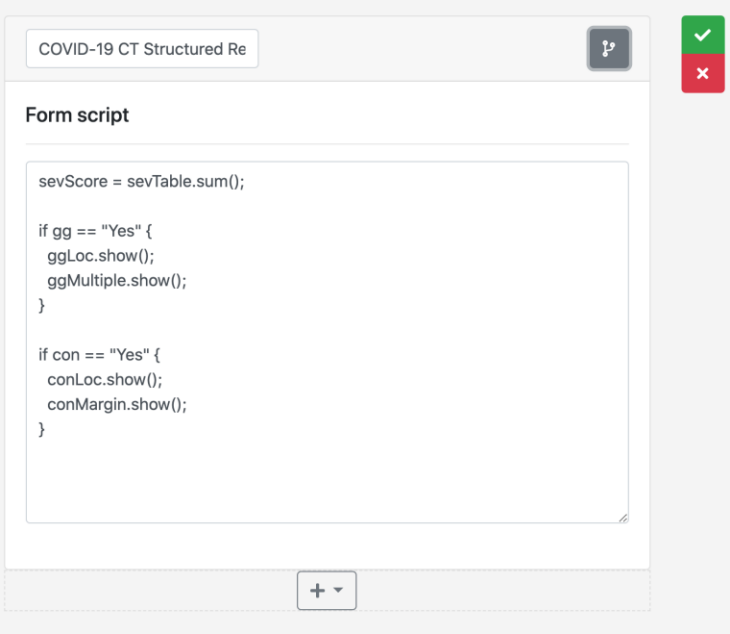

Figure 3 The FormScript editor

4. Impact

With XReport we built a free, cross-platform structured reporting platform for radiologists. It enables both creating and viewing reporting templates in an easy, user-friendly way. We built our software with modular design in mind and refactored the core features into a separate library to make embedding it into other products easy. We also built an application with the library embedded in it to demonstrate the easy integration. Furthermore, we designed a simple DSL called FormScript to add dynamic logic to our forms. The main feature of it, and the reason we created it in the first place, is security. It does not allow malicious code executions unlike the eval function of JavaScript. It is also very simple to use because of its limited feature set. Our templates are dynamic, responsive and have modern design. The templates generate easy to copy-paste structured textual output to be compatible with any HIS, and to integrate well into dictation-based workflows. Our templates help not only in precise reporting, but also serve as a guide for radiologists thanks to our custom form elements such as images and rating tables.

We compared our solution to a similar free service developed by RSNA. From a technological point of view both programs are similar since they are built using web technologies but they have their differences when it comes to the ecosystem, editing process and user experience. The RSNA template library has a more mature ecosystem: there are a lot of contributors who build and upload templates, there are some nice to have features such as favouriting a template. But the template editing itself is less advanced than ours. In the RSNA editor the screen flow to get to the actual editing is as following: click on “Create and Upload a Template button”, click on “T-Rex Template Editor”, interact with a popup which asks how the user wants to start the editing, click on one of the options. In our program the screen flow is a lot simpler: click on “Add new template”, and you are in the editor. In the RSNA editor adding individual elements has some issues. The elements have to be drag and dropped from a side panel, which is problematic on mobile devices as there is not enough space. The element editor works as a pop-up which brings the user out of the editing context. In our app adding elements is responsive (works on mobile devices as well), and is inline, so the user remains in the editing context throughout the whole process. When it comes to how dynamic the templates are we found that RSNA templates do not allow dynamic behavior such as hiding/showing elements based on certain conditions. Through FormScript our system enables fully dynamic behaviour. The RSNA editor lacks some important elements such as images and rating tables which are essential in information sharing and oncological grading systems.

5. Conclusions

This paper introduces XReport, a free, web-based structured reporting platform for radiologists. It enables both creating and viewing reporting templates in an easy, user-friendly way. Our system is deployed at https://app.radiosheets.com and is ready to be used. Template creation and editing requires login, but template viewing, copying the generated reports and sharing the templates do not. The templates currently available in the app have been created and are used by our research group and by radiologists from Pécsi Diagnosztikai Központ and from the University of Pécs.

Conflict of Interest

- *No conflict of interest exists:*
  We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

A. T. was supported by the Bolyai Scholarship of the Hungarian Academy of Science.

References

[1] D. Ganeshan, et al., "Structured Reporting in Radiology.," *Academic Radiology,* vol. 25, no. 1, pp. 66-73, 2018.

[2] D.S. Rana, et al., "Voice recognition for radiology reporting: is it good enough?," *Clinical Radiology,* vol. 60, no. 11, pp. 1205-1212, 2005.

[3] T. A. Morgan, M. E. Helibrun and C. E. Kahn, "Reporting Initiative of the Radiological Society of North America: Progress and New Directions," *Radiology,* vol. 273, no. 3, pp. 642-645, 2014.

[4] C. Simonyi, M. Christerson and S. Clifford, "Intentional software," in *ACM Sigplan Notices*, 2006.

[5] N. Jain, A. Bhansali and D. Mehta, "AngularJS: A modern MVC framework in JavaScript," *Journal of Global Research in Computer Science,* vol. 5, no. 12, pp. 17-23, 2014.

Current executable software version

Ancillary data table required for sub version of the executable software: (x.1, x.2 etc.) kindly replace examples in right column with the correct information about your executables and leave the left column as it is.

Table 2 – Software metadata (optional)

Nr	(Executable) software metadata description	***Please fill in this column***
S1	Current software version	*1.4.1*
S2	Permanent link to executables of this version	*https://app.radiosheets.com*
S3	Legal Software License	*MIT License*
S4	Computing platforms/Operating Systems	*Cross-platform, Web-based system*
S5	Installation requirements & dependencies	*No installation needed, works in any modern browser.*
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	
S7	Support email for questions	ahmedharmouche92@gmail.com