Ελληνικό Μεσογειακό Πανεπιστήμιο

Σχολή Μηχανικών

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πτυχιακή εργασία

Τίτλος:

Ανάπτυξη συστήματος δημιουργίας αναφορών για το πληροφοριακό σύστημα SYNTHESIS (Report writing subsystem for SYNTHESIS Information System)

Πατεράκης Εμμανουήλ (Α.Μ. : ΤΠ4319)

Επιβλέπων καθηγητής : Βιδάκης Νικόλαος

Επιτροπή Αξιολόγησης : Παπαδάκης Νικόλαος
Βιδάκης Νικόλαος
Κονδυλάκης Χαρίδημος

Ημερομηνία παρουσίασης: 25/02/2021

Acknowledgements

I would like to express my appreciation to my supervisor Dr. Nikolaos Vidakis and my sincere gratitude to Dimitris Angelakis for their immense support as well as providing me with the chance to be part of the CCI lab. In addition, I would like to express my utmost gratitude to Chryssoula Bekiari. I wish to extend my special thanks to Corina Doerr for her crucial contribution to the Design process of this project. I am also wholeheartedly thankful to all members of ISL-FORTH's CCI lab.

Abstract

Templatized reporting is a report generation technique utilizing template files for exporting reports. Each report template provides a customized report formatting and stylization, allowing more flexibility of the generated reports' layout. Ontology based data systems have no generic solution for exporting their data in reports. Users of the ontology-based system SYNTHESIS required automated report exporting of the system's existing data in specific and distinct formatting which derived from each report type. The automated exporting of such reports would drastically improve the efficiency of result analysis. A templatized report generator webapp was designed and implemented. The developed tool generates and uses template files in JSON format, each containing the structure of an ontology as that is found in the system's database, alongside all the custom formatting data for the template. Users can create new templates or edit existing ones in order to suit a report type's layout. After a template is created and edited, it can be used to export one or more ontology instances' data in report form by combining the data of each selected instance with the selected template's layout and formatting. The generated report is previewed and then downloaded in the desired format. The developed program was finally tested in a testing deployment and it was determined that it fulfills the initial requirements for SYNTHESIS' report generation.

Keywords: webapp, report, template, PDF, DOCX, XLSX, XML, XQuery, JSON

Περίληψη

Η δημιουργία αναφορών βάσει προτύπων είναι μία τεχνική παραγωγής αναφορών που αξιοποιεί αρχεία προτύπων για εξαγωγή αναφορών. Κάθε πρότυπο αναφοράς παρέχει μια προσαρμοσμένη μορφοποίηση και διαμόρφωση, επιτρέποντας μεγαλύτερη ευελιξία στη διάταξη των παραγόμενων αναφορών. Συστήματα βασιζόμενα σε οντολογίες δεν έχουν μία γενική λύση ώστε να εξάγουν τα δεδομένα τους σε αναφορές. Χρήστες του βασισμένου σε οντολογίες συστήματος SYNTHESIS είχαν απαίτηση για αυτοματοποιημένη εξαγωγή αναφορών των δεδομένων του συστήματος σε καθορισμένη και διακριτή μορφοποίηση που προέρχεται από κάθε τύπο αναφοράς. Η αυτοματοποιημένη εξαγωγή αυτών των αναφορών θα μπορούσε να βελτιώσει δραστικά την αποδοτικότητα της ανάλυσης αποτελεσμάτων. Μία δικτυακή γεννήτρια αναφορών βασισμένη σε πρότυπα σχεδιάστηκε και υλοποιήθηκε. Η αναπτυγμένη εφαρμογή παράγει και χρησιμοποιεί αρχεία προτύπων σε μορφή JSON, όπου το καθένα περιέχει την δομή μίας οντολογίας όπως αυτή βρίσκεται στη βάση δεδομένων του συστήματος, παράλληλα με όλα τα δεδομένα μορφοποιήσεων για το πρότυπο. Οι χρήστες μπορούν να δημιουργήσουν καινούρια πρότυπα ή να επεξεργαστούν τα υπάρχοντα ώστε να ταιριάζουν τη διάταξη μιας μορφής αναφορών. Εφόσον ένα πρότυπο έχει δημιουργηθεί και επεξεργαστεί, μπορεί να χρησιμοποιηθεί ώστε να εξάγει τα δεδομένα από ένα ή περισσότερα στιγμιότυπα (instances) κάποιας οντολογίας σε μορφή αναφοράς, συνδυάζοντας τα δεδομένα κάθε επιλεγμένου στιγμιότυπου οντολογίας με τη δομή και μορφοποίηση του επιλεγμένου προτύπου. Η παραγόμενη αναφορά προεπισκοπείται και μπορεί να κατέβει στην επιθυμητή μορφή αρχείου. Η αναπτυγμένη εφαρμογή δοκιμάστηκε σε μία δοκιμαστική εγκατάσταση και καθορίστηκε ότι πληροί τις αρχικές προϋποθέσεις για την παραγωγή αναφορών του SYNTHESIS.

Keywords: webapp, report, template, PDF, DOCX, XLSX, XML, XQuery, JSON

Table of Contents

Acknowledgements 1
Abstract 2
Περίληψη 3
List of Acronyms 6
List of Figures 7
1. Introduction 8
 1.1. Motivation 8
 1.2. Purpose 8
 1.3. Initial Problem Statement 8
 1.4. Outline 9
2. Pre-Analysis - State of The Art 10
 2.1. Templatized Report Engines 10
 2.2. Non-Templatized Report Engines 10
 2.3. Final Problem Statement 11
3. Methodology 12
 3.1. Method 12
4. Analysis 13
 4.1. Software requirements 13
 4.2. Report Generators 13
 4.3. Templatized Reporting 13
 4.4. WYSIWYG Applications 14
 4.5. XML and XPath 14
 4.6. SYNTHESIS System Overview 15
5. Design 16
 5.1. Software Libraries Overview 16
 5.2. Libraries licensing 19
 5.3. System Architecture 20
 5.4. Preliminary Use Cases 21
 5.5. User Interface Mockups and Workflow 24
  5.5.1. Template Creation Workflow 24
  5.5.2. Template Editing Workflow 27
  5.5.3. Report Generation Workflow 30

6. Implementation 33
6.1. Software Libraries 33
6.2. Interaction with the system 34
6.2.1. SYNTHESIS System Interaction 34
6.2.2. Template Creation & Manipulation 36
6.2.3. Report Generation 42
7. Conclusion 45
7.1. Future Work 45
Bibliography 46

List of Acronyms

UI	User Interaction
UX	User Experience
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
RTC	Report Template Creator
PDF	Portable Document Format
XML	eXtensible Markup Language
SQL	Structured Query Language
DOM	Document Object Model

List of Figures

Figure 1: The Engineering Design Process 12
Figure 2: The materialize logo 16
Figure 3: Demonstration of the Materialize library 17
Figure 4: Demo page of html-docx-js 17
Figure 5: The JSZip demo page, with a complete code example 18
Figure 6: LZMA-JS demo page with output 19
Figure 7: Component Diagram of RTC 20
Figure 8: SYNTHESIS Use Case diagram 21
Figure 9: RTC Use Case diagram 22
Figure 10: Template Creation Diagram 24
Figure 11: Mockup depicting the "Report Templates" submenu entry 25
Figure 12: Mockup depicting the "report Templates" submenu 25
Figure 13: Mockup depicting the form presented when creating a new template 26
Figure 14: Mockup depicting the dropdown list populated 26
Figure 15: Template Editing Diagram 27
Figure 16: Mockup depicting the editing mode of RTC 28
Figure 17: Mockup depicting a finalized updated report template, ready to be saved 28
Figure 18: Mockup depicting the Text Formatting menu of RTC 29
Figure 19: Mockup depicting the image component formatting menu 29
Figure 20: Report Generation Diagram 30
Figure 21: Mockup depicting an ontology's submenu, with the "Export Report" button 31
Figure 22: Mockup depicting the report export form 31
Figure 23: Mockup depicting the dropdown list of templates, populated 32
Figure 24: Mockup depicting the file grouping selection for multiple exports 32
Figure 25: The SYNTHESIS index page 34
Figure 26: The "Report Templates" submenu in SYNTHESIS 35
Figure 27: The report export form 35
Figure 28: The ontology dropdown 35
Figure 29: The template creation's confirmation message 36
Figure 30: Selecting a template for editing in the "Report Templates" submenu 36
Figure 31: A newly created (empty) report template 37
Figure 32: Search functionality 37
Figure 33: The text formatting menu 38
Figure 34: The image formatting menu 38
Figure 35: Template header image 39
Figure 36: A linking button in the field tree 40
Figure 37: The linking form 40
Figure 38: Example of an inserted link 41
Figure 39: Preview of a customized template 41
Figure 40: Selecting an ontology instance for report export 42
Figure 41: The report export form 43
Figure 42: The final report, previewed in RTC in export mode 44

Snippet 1: LZMA compression snippet, returning a compressed string as a promise 39
Snippet 2: Decompressing and inserting another template as link 40

1.Introduction

Ontology based data structures are complicated and introduce difficulties when it comes to gathering results. A severe lack of open-source report generators for ontology-based data has been observed. This project describes the development of an open source templatized report generator for ontology data stored in a structured markup language such as XML or JSON, that aims to be simple and intuitive to operate and implement in other projects.

1.1. Motivation

Ontologies have become a standard for storing research data in the recent years. A system which benefits from the use of ontologies is SYNTHESIS [1], an open-source application that provides functions for storing, editing, and traversing scientific data, developed by ICS-FORTH. The SYNTHESIS user base required the stored data to be exported in the form of reports in order to optimize the result analysis process. This requirement was supported by the desire to create a generic solution, and not one solely for the SYNTHESIS system. Such a solution would be beneficial for a vastly larger number of potential users.

Motive for this project was the need for generating reports in SYNTHESIS using a generic solution in order to benefit a larger user base, a task which had no clear candidate software for this requirement.

1.2. Purpose

The main purpose of this project is to create an easy to operate open-source tool which can generate reports of ontology data, either individually or en-mass, by utilizing templates. The resulting program aims to help researchers export the desired data in a quick and simple manner, making result gathering faster and more efficient. The program was purposely developed as open source to provide as much inclusion as possible by removing the concerns of licensing.

1.3. Initial Problem Statement

Exporting data from ontology-based systems is vital for evaluating results and coming to conclusions. It can however prove a highly complicated task. A report generator for ontology data was required, but no open-source solution was available to fulfill this requirement. Although templatized report generators already exist, they are either not designed for ontology data, not open source, or neither of the two.

1.4. Outline

The starting chapter of this project is Introduction where the motivation, purposes and initial problem statement are defined. Chapter 2 includes the background research that was conducted and a finalized problem statement, which was redefined based on that research. Chapter 3 describes the methodology that was followed to complete this project. Chapter 4 contains the project requirements, for both software libraries, and mechanisms. Chapter 5 describes the design process leading to the final user interface. Chapter 6 presents the implementation which derived from the design process and resulted in the final program.

2. Pre-Analysis - State of The Art

Reporting engines are software tools designed to generate report files based on existing data. Exporting existing data in varying formats and file types is one of the main reasons reporting engines are implemented in applications with large databases.

2.1. Templatized Report Engines

Templatized report engines allow for faster and more uniform report creation, by following a template's formatting and basing all exported reports on the same style principles.

iText [2] is a widely used, largely open-source PDF generating library written in Java. iText focuses on industrial use and provides top-of-the-line pdf creation functionalities. iText also provides a complete template editor, iText DITO [3], which focuses on accessibility, as it requires minimal coding knowledge. While iText is powerful and includes a lot of features, it is licensed under AGPL, with iText DITO being completely closed-source, both of which severely limit the potential user base.

DynamicReports [4] is an open-source Java reporting library based on JasperReports. It allows to create dynamic report designs and it does not need a visual report designer. Instead, it uses JRXML template files, which include the formatting and each report can be generated based on a JRXML file. DynamicReports supports PDF, DOCX, and EXCEL files. DynamicReports is licensed under LGPL-3.0.

2.2. Non-Templatized Report Engines

Non-templatized report engines can generate reports manually, without the use of templates. That makes them less ideal than templatized report engines for exporting files from a database, but their advantages lie in more specific functionalities.

jsPDF [5] is an open-source report engine written in JavaScript. The fact that it is a JavaScript library can lead to a better user experience and implementation, as there is no need for any backend implementation; all of jsPDF's usage is taking place on the front end. jsPDF is licensed under the MIT License.

html2pdf [6] is an open-source JavaScript library that converts HTML files (alongside their CSS counterparts) or elements into PDF reports, using html2canvas [7] and jsPDF. The resulting PDF files are perfect copies of the given HTML files, but they are converted to images and hence contain no text elements. This makes the library perfect for 1:1 exporting of HTML files, though it is problematic for extracting readable data from the final reports. Html2pdf is licensed under the MIT License.

Openhtmltopdf [8] is an open-source Java library for rendering XML/XHTML or HTML files using CSS 2.1 for layout and formatting, outputting to PDF or images. This library allows for the creation of almost perfect PDF clones of HTML files, by converting each HTML element into a respective PDF one, keeping the text nature of the element including all the original formatting and style. Openhtmltopdf is licensed under LGPL.

2.3. Final Problem Statement

Data evaluation for ontology-based systems requires exporting the stored data in a readable form for the according experts to analyze them and come to conclusions. Exporting ontology data to readable forms and more specifically reports is achieved through the use of report engines. Report engines are tools designed to generate reports based on a given dataset using a predefined report format.

After evaluating the most prominent templatized and non-templatized report engines it is evident that none of the existing tools provide the functionality required to export ontology-based data to reports. Therefore, a report generator needs to be developed which will enable ontology data to be seamlessly exported to reports. Since templatized report generation is faster, simpler, and more efficient when it comes to creating similarly-styled reports, the resulting program should also be a templatized report generator, allowing users to create and use different templates. The developed tool can be based on an existing report generator, granted the latter would have to be heavily modified or used simply as a library in a separate, standalone program. A combination of report engines could be incorporated for this project in order to support multiple report generating functions and file formats for better compatibility.

Problem Statement:

"How can an open source templatized report generator be developed to assist with ontology data evaluation?"

3. Methodology

Computer programs are developed based on source code generation, accompanied by multiple code-oriented assisting, reporting and evaluation tools. Software Development Methodology describes the development process and how it is managed, to help assure that a project can be completed when expected, without exceeding the time (and therefore monetary) budget. This methodology can differ between projects, depending on the type of the final program, its usage, the tools utilized for its development and more.

The development method that was selected for this project is the Engineering Design Process [9]. This method is based on a series of steps that describe the development process leading up to the creation of a new product or system. This process is not considered linear, engineers often move back to previous steps, and reiterate them in order to reach a final solution.

3.1. Method

The Engineering Design Process steps are as follows:

1. Define the problem
2. Do background research
3. Redefine the problem
4. Specify requirements
5. Brainstorm solutions, choose the best solution and develop it
6. Build a prototype
7. Test and redesign
8. Communicate results

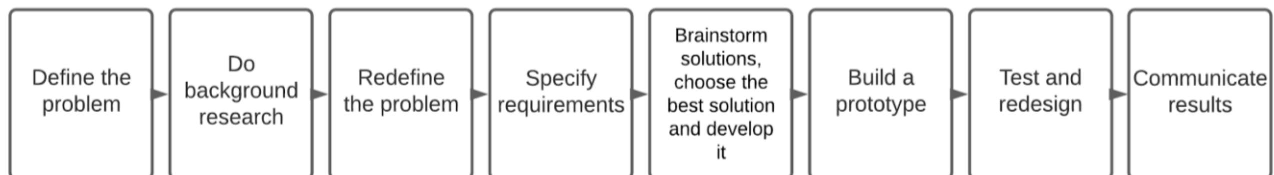


Figure 1: The Engineering Design Process

The aforementioned steps were followed to research, design, implement and improve this project. Initially the problem was defined and specified. This allowed background research to be conducted in order to get a deeper understanding of the challenges and demands of the project, followed by redefining the problem according to the newly gathered information. Next, the project requirements were specified. A plethora of solutions were brainstormed and presented based on the final problem definition and the project requirements. The best solution was selected, and an early design was created, describing all the possible different workflows. A prototype program was developed based on the design. The prototype was evaluated and redesigned multiple times based on developer, designer and user feedback, until it reached a satisfying form.

4. Analysis

4.1. Software requirements

The resulting program was required to be an open-source Java Servlet project, compatible with XQuery/XPath, more specifically eXist-db. The project had to be distributed as a Maven package, while being fully compatible with the EUPL-1.2 license.

Apache Maven [10] is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. A Maven engine is required for acquiring the necessary libraries, even when the program's basic source code is provided.

Java Servlets are web-oriented Java classes which are used for handling the program's API, hence the Java runtime is another requirement. The Java servlets provide a way for the webapp to transfer data to and from the database.

eXist-db [11] is an open-source xml-based database, which supports XQuery and XPath. It is being used on the SYNTHESIS system and is implemented for this program as well, in a manner similar to that of SYNTHESIS in order to provide total compatibility.

An HTTP server is also required for deploying the webapp on the web, or locally.

4.2. Report Generators

Report generators [12] are computer programs that allow users to directly retrieve all the data they want from a database, spreadsheet, XML stream, or any other source, and view it online or export it to different document formats such as Excel, PDF, and CSV that satisfy a specific human readership. Report generators are using report engines to generate a report based on a given dataset.

Report generation eliminates the need for manually creating reports, lowering the risk of errors, and allowing better data analysis. It uses a report generator, a tool that requires defining the report, including the type of data to retrieve, the location of data, and the method of displaying it. This allows users to create reports based on existing information (for example, a database) with the given report definition and combining them with the report layout to produce the report.

4.3. Templatized Reporting

Templatized reporting [13] is a report generation method that utilizes user-created templates to output the final report(s) based on the template's properties. Instead of using a static style, layout and data, user-created templates can be customized to suit specific purposes. Furthermore, templates can be used to create reports for similar purposes using different datasets for each report.

To fulfill a report generation request, a template must be first created and selected, alongside some basic data, then the template is combined with data to create the final report. To create the template, a report (template) creation request is created by the user, and a basic template is created by the system based on the received request and its identified data. A data generator will be later used to generate the final output based on the created template. To output the final report, a report output request is created, where a template and some basic parameters are selected by the user. The report generator then retrieves the template and all the required data based on the given parameters, to finally insert all the data into the template. The report is at this point ready for output, that can be either displaying the final result on the user's screen or saving it as a file.

4.4. WYSIWYG Applications

WYSIWYG stands for "what you see is what you get", which is a programming principle where the output is derived from the preview, becoming a near perfect clone of the latter. This principle is used in applications that implement an editing environment similar (or identical) to the finished product, which leads to users essentially creating the end result itself through the UI. Examples of WYSIWYG applications are text editors like Google Docs [14] which can export files identical to the ones previewed in the editing environment, but also assisted/automated web page engines like WordPress [15], where users can edit the final page layout and design through the editing UI.

4.5. XML and XPath

XML [16] is a software-independent markup language used for storing and transporting data. XML is also extensible (XML is an acronym for eXtensible Markup Language) as it supports custom tags and attributes instead of predefined ones. Its software independence is achieved using the features mentioned above but also because XML stores data in plain text format. Consequently, ontology-derived XML Schemas can be used to create XML mappings to maintain a compliance between an ontology and a generated XML document. Additionally, XML allows for easier expansions or upgrades to new operating systems and applications without data loss, while also providing interoperability between different systems.

XPath is a syntax for defining parts of an XML document. It is used to traverse XML documents and select nodes or node-sets within the XML structure. More specifically, it can be used to navigate through elements and attributes in an XML document through path expressions. The path expressions used by XPath are very similar to the expressions used in traditional computer file systems. XPath can be used in the most widely used programming languages such as JavaScript, Java, XML Schema and PHP. XPath is a major element in the XSLT standard and XQuery.

XML datasets can be implemented either in raw XML files or in the form of databases that use XPath (and XQuery) for data gathering requests. This is a type of NO-SQL database, where there is no need for using SQL, instead the database can rely solely on the XML and XPath structure and mechanisms.

4.6. SYNTHESIS System Overview

SYNTHESIS is a management information system developed by the Center of Cultural Informatics of ICS-FORTH in collaboration with multiple cultural institutions worldwide. It employs the CIDOC CRM for designing the data structures which are stored in XML (and XSD) files. CIDOC CRM is an ontology standard, recommended as the most appropriate for documenting cultural entities. The XML files derived from CIDOC CRM stored are managed with eXist-db.

The system's initial goal was to provide the administration and scientific documentation procedures, while supporting multiple languages, data exchange and interoperability. Later implementations added e-services designed to inform target audiences (such as teachers, researchers, schoolchildren, and the general public) about different monuments, historical persons, events etc. This cultural information is stored in the SYNTHESIS database as XML documents that are described by XML schema. Users with the role of editor can modify the content of these documents using an XML editor, while lower privilege users can only view the same content.

Additionally SYNTHESIS supports a generic workflow for documenting cultural entities, allows the creation, editing, navigation and retrieval of documents, supports data migration, document translation, import from and export to XML/RDF and associating documents of the primary cultural entities with documents of additional secondary entities.

5. Design

In this project a web application (web app) was designed to provide the functions of a complete report generator. The design was purposely devised to be compatible with ontology-based data, both in its appearance and function.

Since each ontology has a determined structure (in the form of XML schema), a template mechanism was proposed in order to provide similar report types for varying datasets of the same ontology (or ontologies). This mechanism uses the application's report template creator to create and edit each report template; this is why the program was given the name Report Template Creator or RTC. Templates are used for customizing the ontology's components and eventually help generate the final reports based on this customization.

The application is intended to function on any ontology-based system, it is therefore purposely designed as a plugin. As the purpose of this report is to support report generating for the SYNTHESIS system, the template ontologies are designed as ontologies of the SYNTHESIS system itself.

It is important to note that the initial design mockups were illustrated by Corina Doerr, who is a member of the SYNTHESIS team, although her involvement is outside the scope of this paper.

5.1. Software Libraries Overview

Software libraries were carefully selected after thorough examination of each library's function(s) and comparisons of all suitable libraries against each other. Every candidate library had to be open source, as per the requirements, and fully licensed with a compatible EUPL license. The selected libraries are the following:

- *Materialize* [17] [18]: Materialize is an open-source JS/CSS library which implements the Material Design as suggested by Google. It was selected for this project because it provides the required Material Design Look and Feel, while being very simple to implement.
  The Materialize website states: "By utilizing elements and principles of Material Design, we were able to create a framework that incorporates components and animations that provide more feedback to users. Additionally, a single underlying responsive system across all platforms allow for a more unified user experience."

Figure 2: The materialize logo

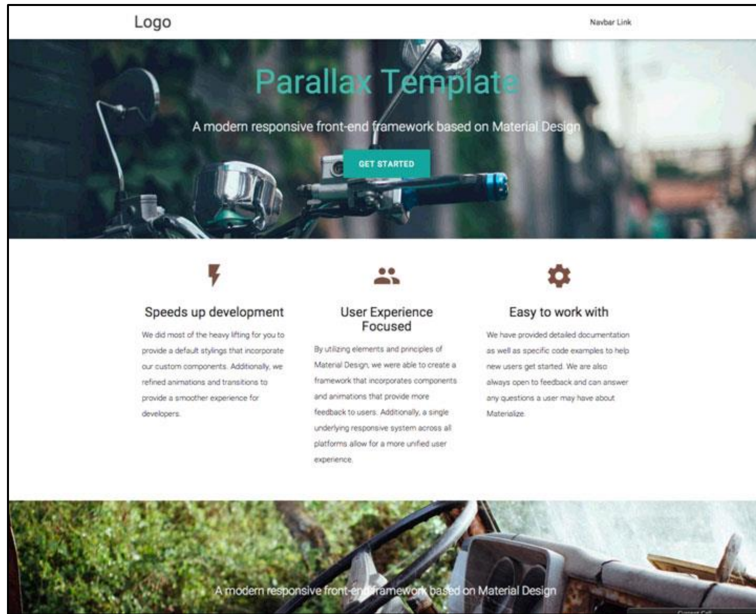


Figure 4: Demonstration of the Materialize library

- *Html-docx-js* [19]: Open-source JavaScript library which provides the ability to convert HTML data in combination with CSS to DOCX format that is used by Microsoft Word. It was selected for this project due to its front-end nature, which nullifies the need for a backend implementation, while providing all the required features for DOCX file creation. The GitHub page states: “This is a very small library that is capable of converting HTML documents to DOCX format that is used by Microsoft Word 2007 and onward. It manages to perform the conversion in the browser by using a feature called 'altchunks'. In a nutshell, it allows embedding content in a different markup language. We are using MHT document to ship the embedded content to Word as it allows to handle images. After Word opens such file, it converts the external content to Word Processing ML (this is how the markup language of DOCX files is called) and replaces the reference.”

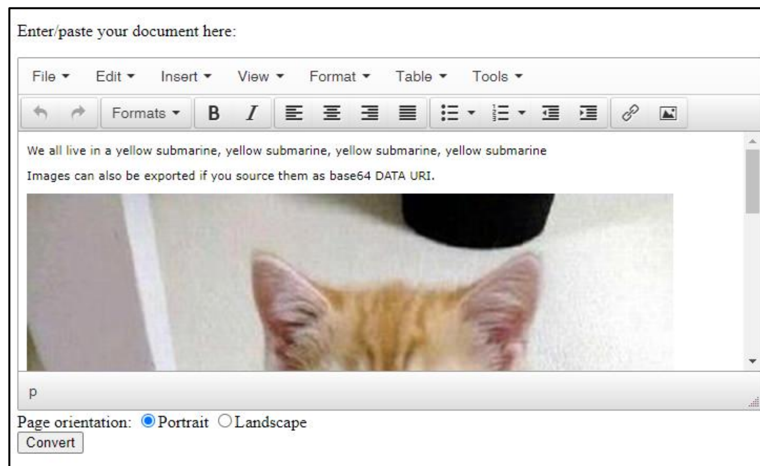


Figure 3: Demo page of html-docx-js

- *Openhtmltopdf*: Open-source library written in pure Java, used for rendering HTML5 and outputting PDF files. It was selected for this project due to its HTML support, alongside its ability to create text-based PDF files (instead of creating PDFs of images). While it has a complicated implementation, it provides all the required features for PDF file creation as defined in the Analysis phase.
  The GitHub page states: “An HTML to PDF library for the JVM. Based on Flying Saucer and Apache PDF-BOX 2. With SVG image support. Now also with accessible PDF support”. “Open HTML to PDF is a pure-Java library for rendering arbitrary well-formed XML/XHTML (and even HTML5) using CSS 2.1 for layout and formatting, outputting to PDF or images.”

- *ExcelJS* [20]: An open-source JavaScript library which provides the manipulation and creation of XLSX files that are used by Microsoft Excel. While not being as simple to implement as the other libraries used in this project, it was selected thanks to its front-end nature while providing all the required features for XLSX file creation.
  The GitHub page states: “Read, manipulate and write spreadsheet data and styles to XLSX and JSON. Reverse engineered from Excel spreadsheet files as a project”. The GitHub page includes an extensive API documentation, which was another factor that led to its selection.

- *JSZip* [21]: An open-source JavaScript library for managing ZIP files. It was selected due to its very easy implementation and front-end nature for creating file packages. As shown in the following figure, all the required commands occupy just a few lines.
  The GitHub page states: “A library for creating, reading and editing .zip files with JavaScript, with a lovely and simple API”.

JSZip

JSZip is a javascript library for creating, reading and editing .zip files, with a lovely and simple API.

Current version : v3.2.0

License : JSZip is dual-licensed. You may use it under the MIT license *or* the GPLv3 license. See LICENSE.markdown.

Example

```
var zip = new JSZip();
zip.file("Hello.txt", "Hello World\n");
var img = zip.folder("images");
img.file("smile.gif", imgData, {base64: true});
zip.generateAsync({type:"blob"})
.then(function(content) {
   // see FileSaver.js
   saveAs(content, "example.zip");
});
```

Run!

Installation

With npm : `npm install jszip`

With bower : `bower install Stuk/jszip`

With component : `component install Stuk/jszip`

Manually : download JSZip and include the file `dist/jszip.js` or `dist/jszip.min.js`

Installed ? Great ! You can now check our guides and examples !

Figure 5: The JSZip demo page, with a complete code example

- *LZMA-JS* [22]: Open-source JavaScript library which is used for compressing and decompressing data by implementing the Lempel-Ziv-Markov (LZMA) chain compression algorithm. It was selected to help lower the file size of report template files, while being easy to implement. LZMA-JS is the easiest library used in this project in terms of implementation, with a total of just two lines for compressing or decompressing an item. The GitHub page states: "LZMA-JS is a JavaScript implementation of the Lempel-Ziv-Markov (LZMA) chain compression algorithm."

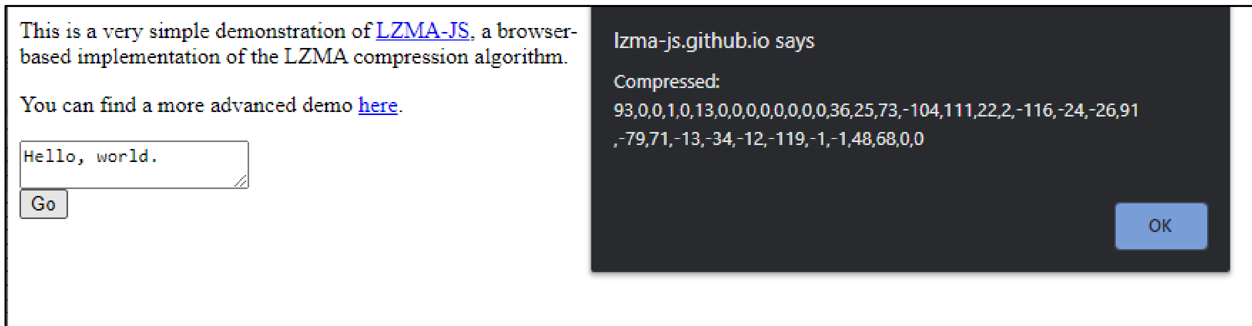


Figure 6: LZMA-JS demo page with output

5.2. Libraries licensing

As mentioned in the Analysis phase, the required license for the project is EUPL. The latest EUPL revision is v1.2 [23], hence all the third-party libraries used in this project are required to be compatible with EUPL-1.2 [24]. Each library's license is listed below:

- *Materialize, Html-docx-js, ExcelJS, LZMA-JS, JSZip**: MIT License
- *JSZip**: GNU General Public License version 3 (GPL-3) [25]
- *Openhtmltopdf*: GNU Lesser General Public License version 2.1 or later (LGPL-2.1)

*JSZip is dual-licensed under MIT and GPL-3, allowing for the use of either license in third party distributions.

5.3. System Architecture

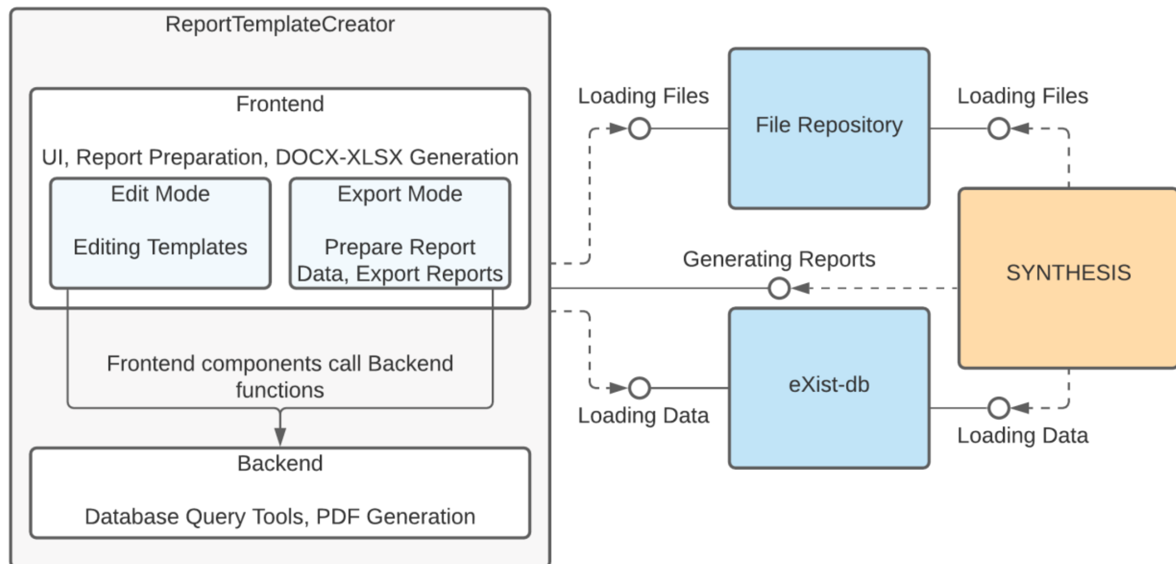


Figure 7: Component Diagram of RTC

RTC is comprised of two main components, its frontend and backend. Each one has different functionality but there is intercommunication between them. Besides its own components, RTC uses two external components, a file repository, and a database, in this case eXist-db. Finally, it used by another external component, SYNTHESIS.

In more detail, RTC's frontend component is responsible for:

- All user interactions, through the incorporation of its user interface, within which a user can use all RTC functionalities.
- RTC's Edit Mode, which offers the user all the necessary tools to edit and enrich the report template files to suit their needs
- RTC's Export Mode, which prepares a report's appropriate data and files according to the selected template and ontology instance, by using the backend functions. Furthermore, it contains the ability to generate DOCX and XLSX files on the client-side, provides the ability for exporting the final generated reports and finally downloading them. The downloads can be either archives in the form of ZIP files, or individual downloads.

RTC's backend component is responsible for:

- Providing a communication layer between RTC and the database/file repository. To avoid cluttering code, the frontend never queries the database directly, instead it always uses the backend to transfer data to and from the database.

- Generating PDF files, which is part of the backend solely due to a library limitation, as there was no satisfactory client-side library for generating PDF files. These files are requested by the frontend and then sent over for exporting.

Finally, SYNTHESIS is incorporating RTC for generating reports of ontologies from its own database and file repository. The templates, data and files are stored on SYNTHESIS' database and file repository, which must be the same used in RTC through its configuration. This way, the intercommunication between the two systems can be seamless, and all that is needed is for SYNTHESIS to call RTC through its own toolbox.

5.4. Preliminary Use Cases

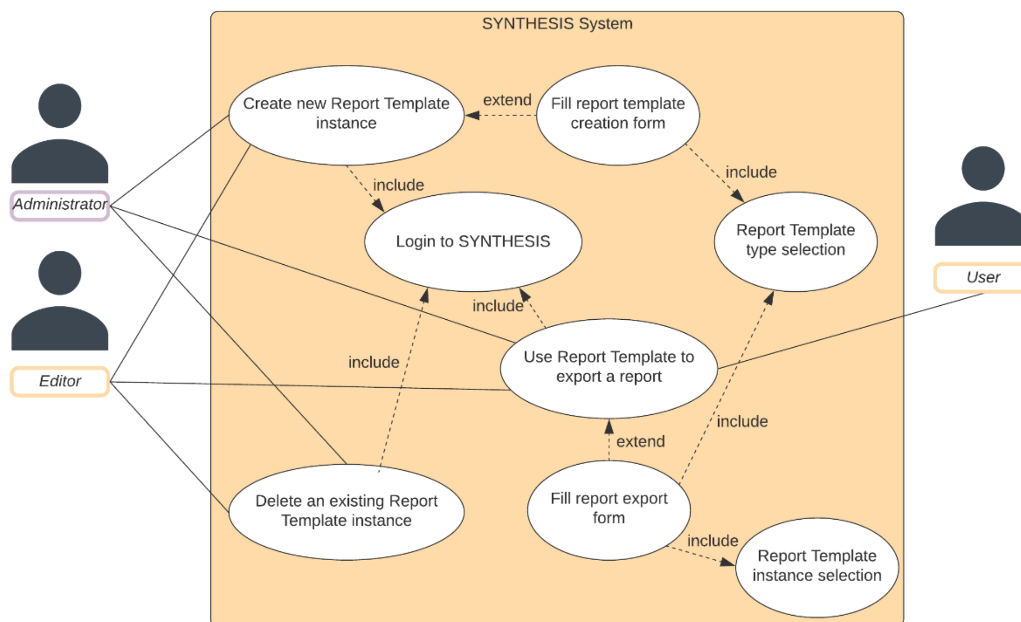


Figure 8: SYNTHESIS Use Case diagram

The diagram above depicts the use cases that take place in the SYNTHESIS system in order to generate a report, starting with the creation of a new template. There are three distinct actors that can partake in this process, the Administrator, the Editor, and the User. The Administrator and the Editor have higher privileges than the User, allowing them to perform more critical actions in the process.

The actions performed by the Administrator and Editor are, in sequential order:

i. Logging in to SYNTHESIS. This is a requirement for all the other actions and for all the users since these actions are taking place in the SYNTHESIS system.

ii. Creating a new report template instance.

a. By extension, filling the report template creation form with all the necessary metadata, including the report template type.

iii. Using the report template in order to export a report. This will cause the RTC window to load in export mode.

a. By extension, filling the report export form with all the necessary metadata, including the report template type and a specific report template instance.

iv. Deleting an existing report template instance from the system. This will cause the instance to be removed from the database and the template will no longer be available to use for generating reports.

The User role is limited to only using existing report template instances that the Administrator and/or Editor have created (as described in iii), in order to export reports using those templates.

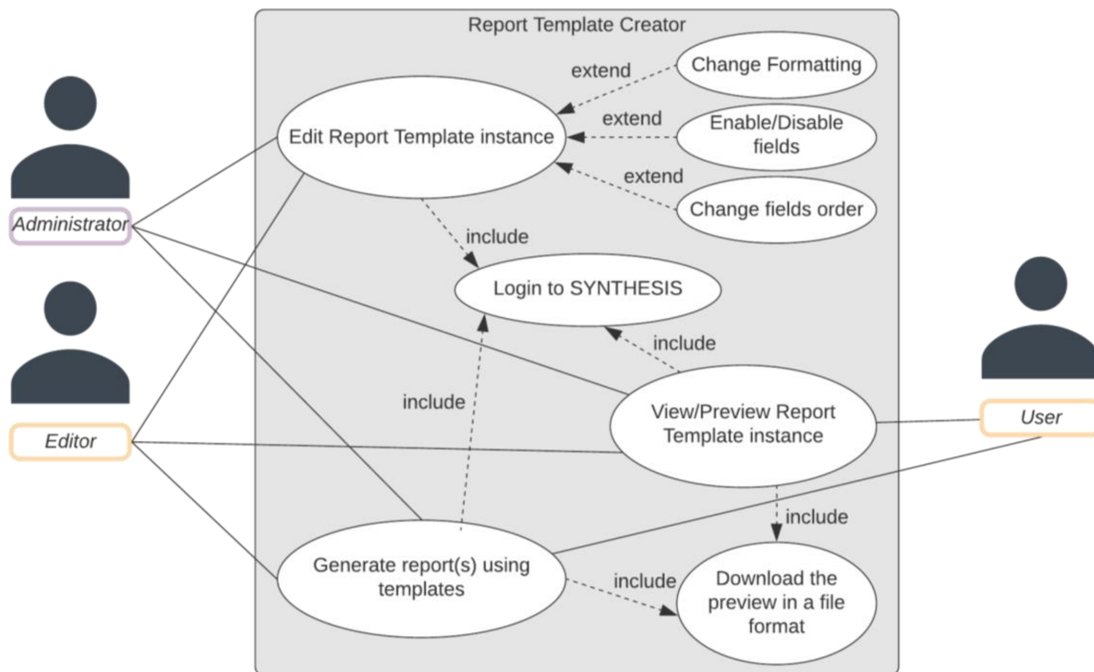


Figure 9: RTC Use Case diagram

The diagram above depicts the use cases that take place in RTC in order to generate a report. These actions are triggered after an action from the previous use-case diagram has been completed. The same three actors from the previous diagram are involved in this process, the Administrator, the Editor, and the User.

The actions performed by the Administrator and Editor are, in sequential order:

i. Same as previously, logging in to SYNTHESIS. This is once again a requirement for all the other actions and for all the users since these actions are based on actions taking place in SYNTHESIS.

ii. Editing a report template instance. This action involves RTC loading in edit mode, allowing the customization and stylization of a template. By extension, the edit mode allows a user to:

    a. Change the formatting of an element group. An element group includes all the elements of the same type, such as field headers. Every element in each group is updated at the same time when a custom formatting is applied to the element group.

    b. Select which fields are enabled. Enabled fields are the only ones visible in all the previews as well as the final report generation.

    c. Change the enabled fields' order. The customized order includes all elements, enabled or not. The re-ordering can only be done between elements of the same depth because the nesting order must be kept intact for XQuery to function properly.

iii. Viewing or previewing a report template. This allows the users to see a preview of the edited template using sample data, in order to provide a representative demonstration similar to the final reports that will be generated using the same template. All the customizations and stylizations are included and being applied in this preview.

    a. The preview function allows users to download the previewed report in their desired file format, such as PDF or DOCX. When a template is being previewed using sample data, the downloaded file will contain the same sample data.

iv. Generating final reports using report template instances. This is the most important part of the process, where the final report is generated by RTC based on the given template and a set of selected ontology instances. This action is a result of action iii from the previous diagram. After generating the final report, RTC displays a preview of the generated report.

    a. Same as in iii, the preview function allows users to download the generated report in their desired file format, such as PDF, DOCX or XLSX. XLSX is used when the report was generated using a Table type template. After downloading the file, the scenarios are completed, as the report export has, at this point, finished.

In this use-case, the User role is limited to only viewing, previewing, and using existing report template instances that the Administrator and/or Editor have edited in order to generate reports using those templates (as described in ii and iii).

5.5. User Interface Mockups and Workflow

5.5.1. Template Creation Workflow

The following workflow diagrams depict the sequence of all the user actions required to create and edit templates, as well as using them for generating reports.

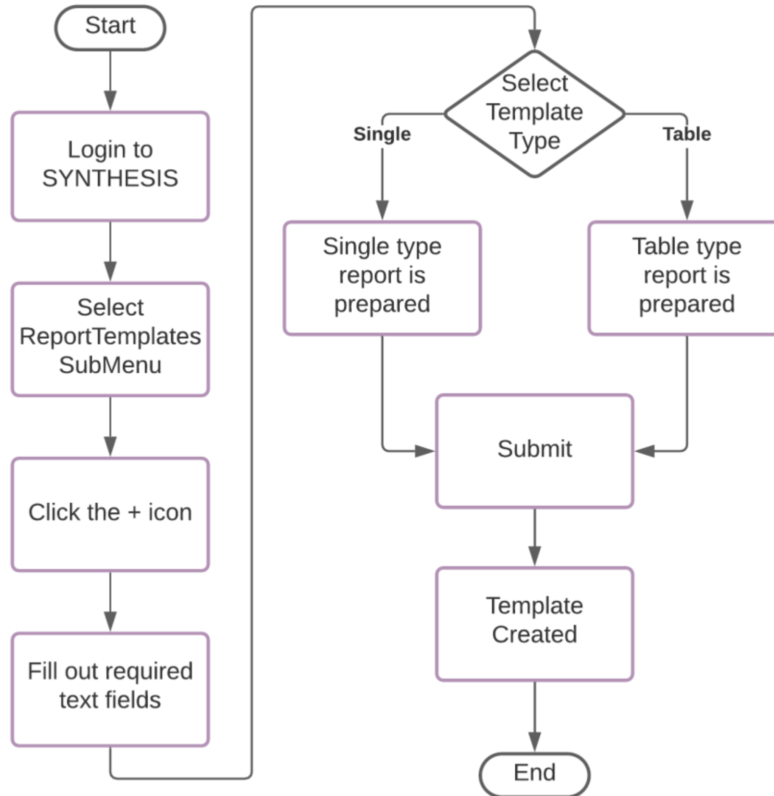


Figure 10: Template Creation Diagram

The above diagram depicts the template creation workflow. This workflow is essentially external to RTC, as the user completes it solely through SYNTHESIS. However, it is still an integral part of RTC's function as presented in this project, hence it is included as such.

First, a user logs in to the SYNTHESIS system using their credentials. After logging in, they are presented with the index page of SYNTHESIS, from which they select the "Report Templates" submenu. This action will cause the "Report Templates" submenu to appear, from where the user clicks on the "+" icon, indicating their desire to create a new report template. Subsequently a form is presented, where they must fill out the required fields and then select the template's type. Finally, they submit the form after which the template is created by SYNTHESIS according to the user provided data and are given the appropriate privileges.

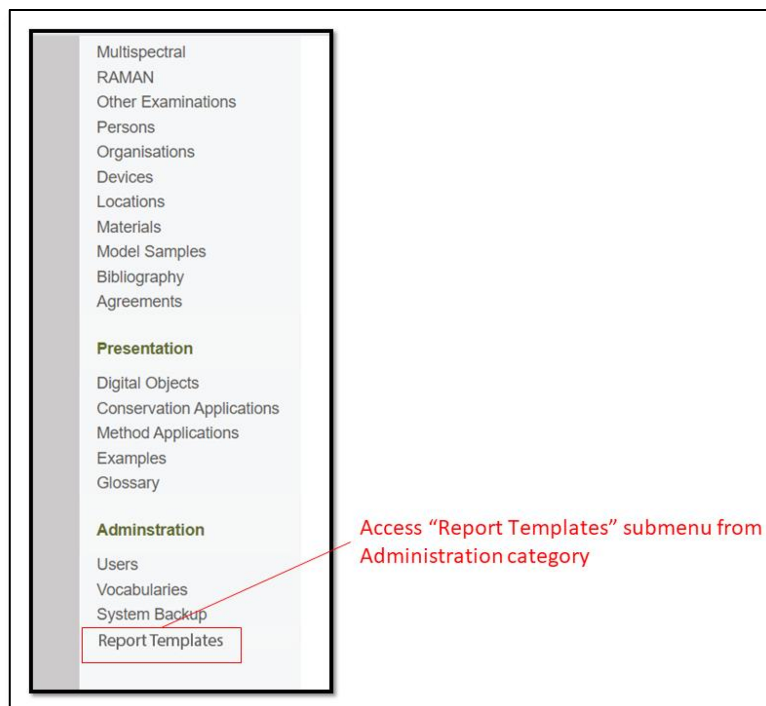


Figure 11: Mockup depicting the "Report Templates" submenu entry

Part of the SYNTHESIS index page, with updated entries to accommodate report templates. The user clicks on the "Report Templates" button and gets redirected to the following figure, the "Report Templates" submenu.

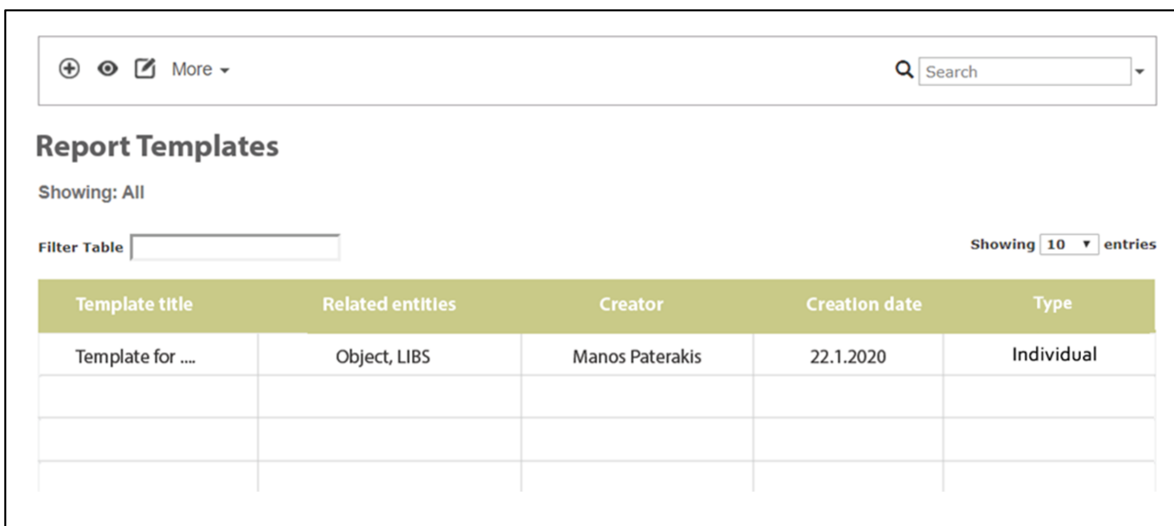


Figure 12: Mockup depicting the "report Templates" submenu

The user is presented with the "Report Templates" submenu and can create a new Template by clicking on the "+" icon placed at the top left.

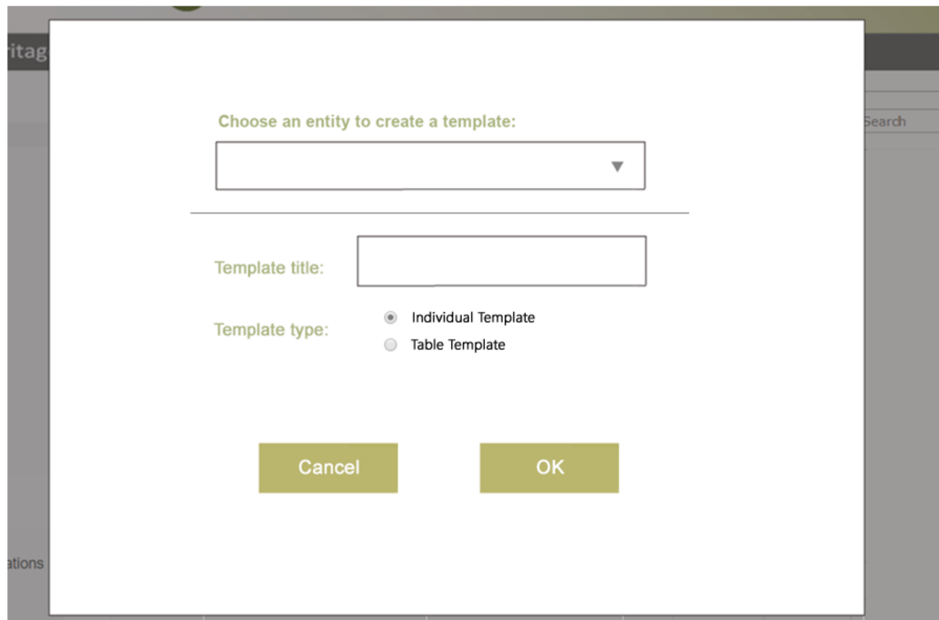


Figure 13: Mockup depicting the form presented when creating a new template

The user is shown this form right after clicking the "+" icon from the previous figure. At this point they must fill out this form by selecting an entry from the dropdown, entering a string text as the title, and finally determining the template's type. After that they press "OK" to create the template.

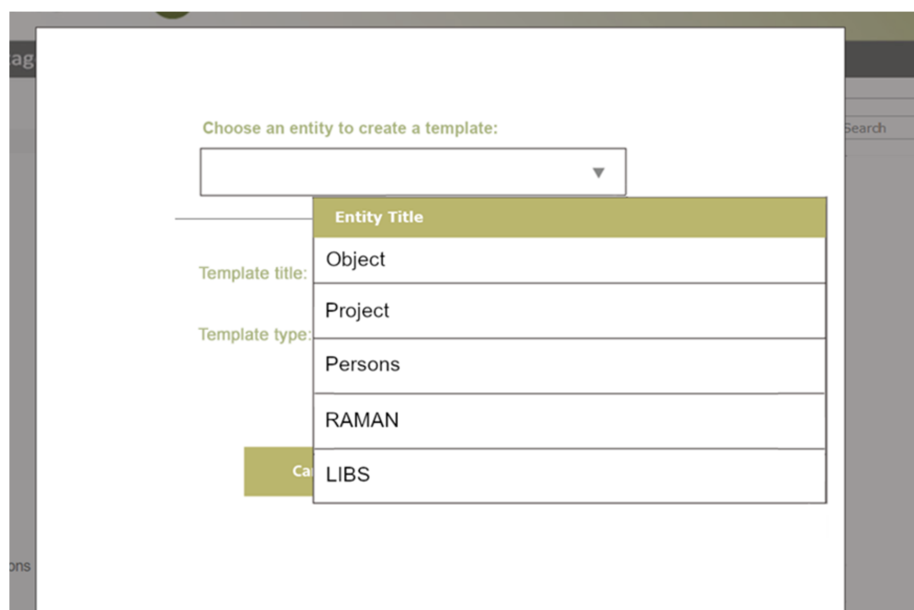


Figure 14: Mockup depicting the dropdown list populated

The dropdown list from the previous mockup, visible and populated with data. All available ontologies of the SYNTHESIS database are visible in this dropdown.

5.5.2. Template Editing Workflow

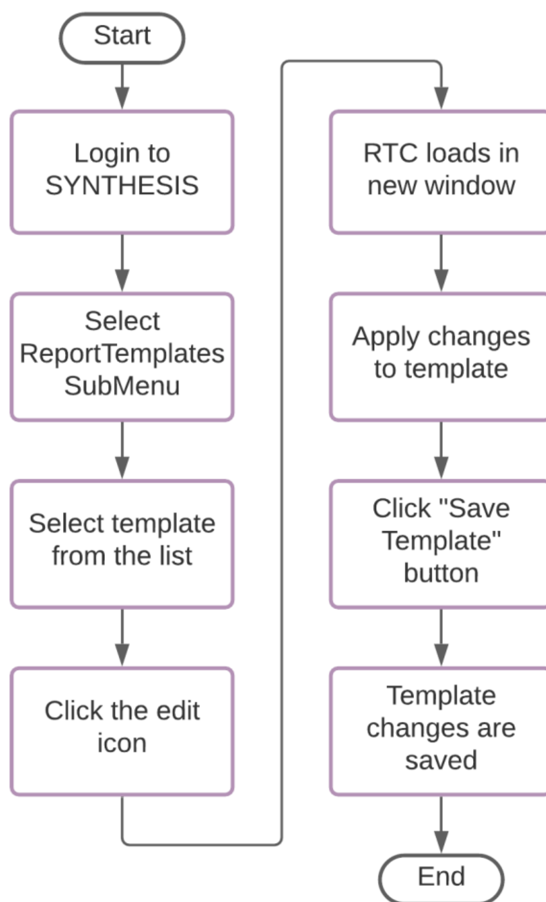


Figure 15: Template Editing Diagram

The above diagram depicts the template editing workflow. Same as previously, the user logs in to the SYNTHESIS system and then they select the “Report Templates” submenu. From within the submenu, they select the template instance they want to edit from the presented list and click the edit icon. This causes the RTC window to appear in edit mode, with the selected template loaded and ready for editing.

After the RTC window is loaded, the user can apply any desired changes to the template; these changes will be presented in more detail in the following figures. After all desired changes have been completed and the template has reached a satisfactory state, the user clicks on the “Save Template” button which will cause all the applied changes are saved in the SYNTHESIS database.

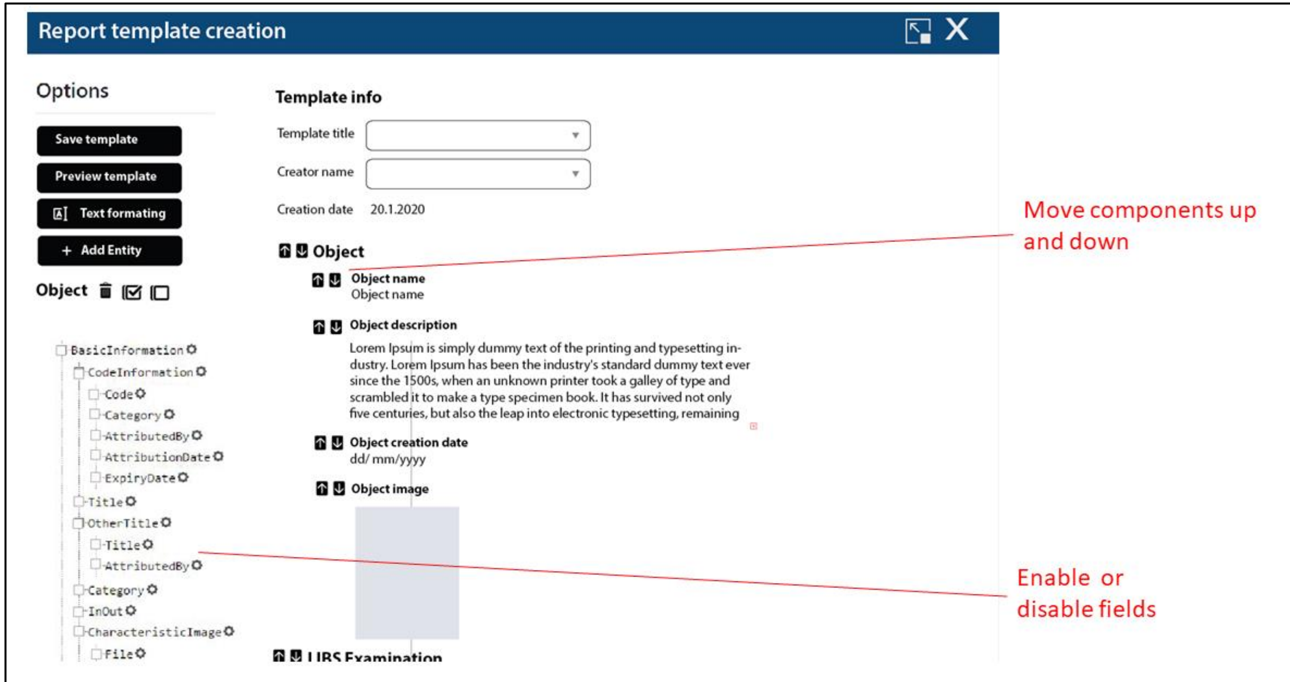


Figure 16: Mockup depicting the editing mode of RTC

The main screen of RTC's editing mode. From here the user can select the items they desire to enable, change positions of each item, edit the template's data such as the title, change text formatting, preview changes, and add external entities.

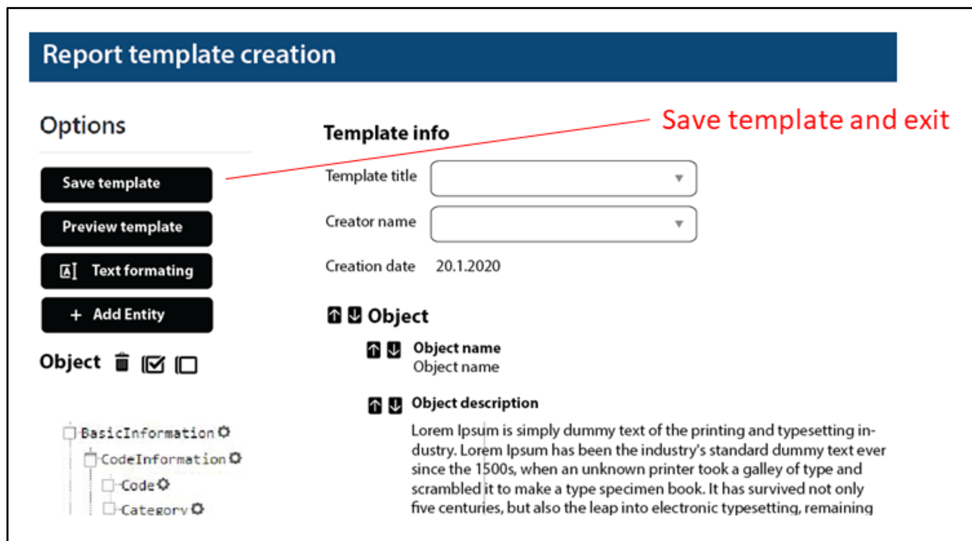


Figure 17: Mockup depicting a finalized updated report template, ready to be saved

After applying all the desired changes, the user must click on the "Save Template" button which will cause the template file to be updated on the SYNTHESIS database.

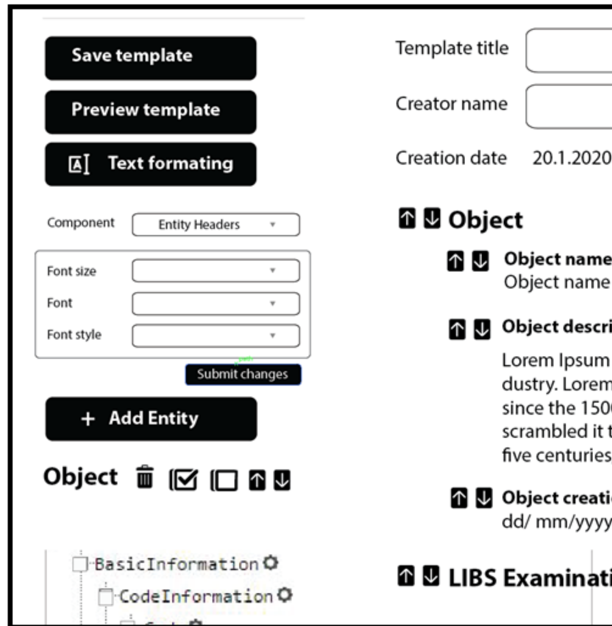


Figure 18: Mockup depicting the Text Formatting menu of RTC

The text formatting menu can be toggled between open and closed by clicking the "Text Formatting" button. In this mockup a basic set of text formatting features is included, such as font, font size and font style, grouped by Component category. The grouping indicates which component the current formatting selection is being applied to. After selecting the desired values, the user can apply the changes by clicking the "Submit changes" button.

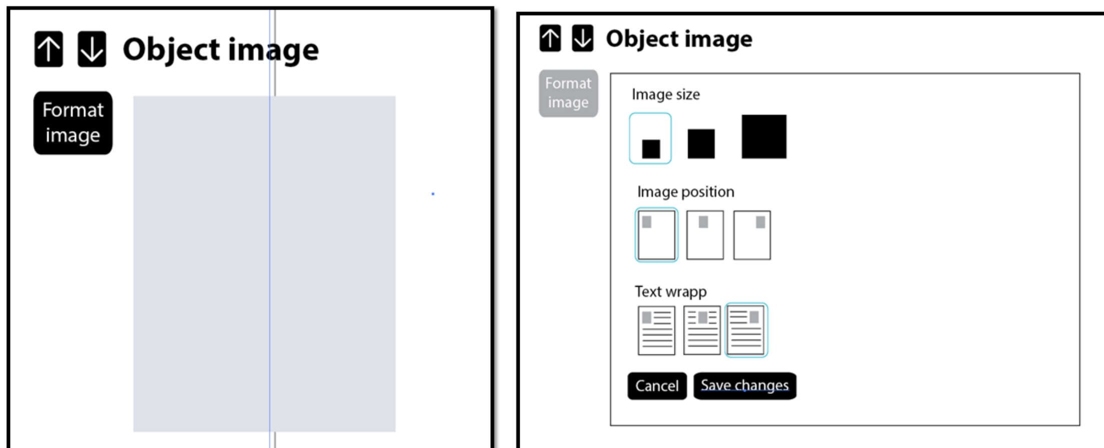


Figure 19: Mockup depicting the image component formatting menu

An image component can have specific formatting which is edited through a specialized menu. This menu can be toggled between visible and invisible by clicking the "Format Image" button, where some basic image formatting tools are presented, such as image size, position, and text wrapping. The changes can be dismissed by clicking "Cancel" or applied by clicking "Save Changes".

5.5.3. Report Generation Workflow

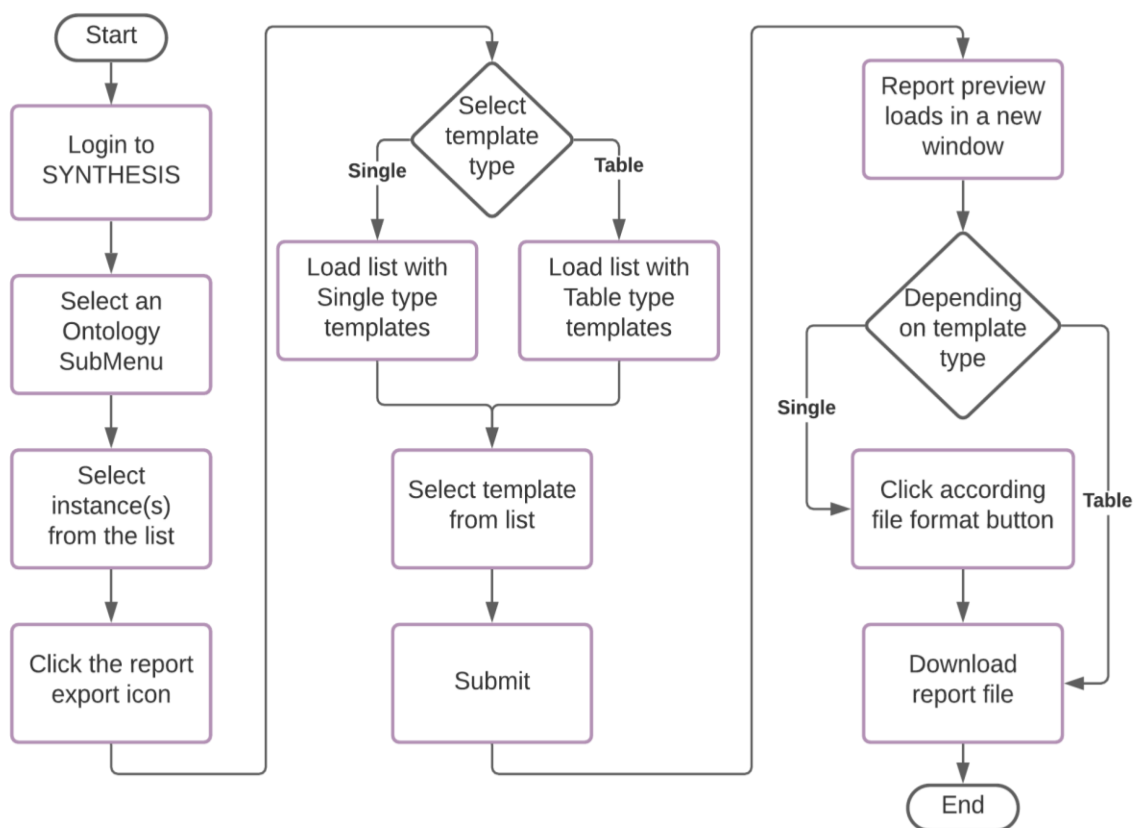


Figure 20: Report Generation Diagram

The above diagram depicts the template generation workflow. First, the user logs in to the SYNTHESIS system. They then select an ontology submenu from the ontologies list. Following, they select an instance from the presented list and click the report export button. Subsequently a form is presented, where they select the desired template type. This causes a list of compatible (with the selected ontology) templates to be loaded, and the user selects the desired template from the list. Finally, the user submits their request for a report generation.

Submitting the form for a report generation will automatically cause the report preview to load in a new window of RTC in export mode. Depending on the template type, the user can either click the format file they prefer, causing the according download to start or wait for the download to start automatically. The former concerns Single type templates and reports, while the latter concerns Table type templates and reports. In both cases, this concludes the report generation workflow as the files have at this point been generated, exported, and downloaded.

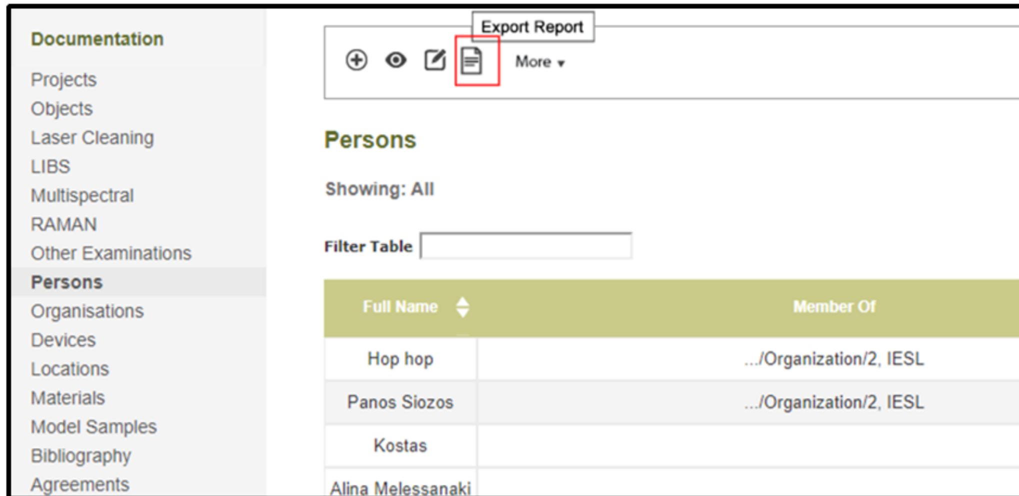


Figure 21: Mockup depicting an ontology's submenu, with the "Export Report" button

After entering an ontology's submenu (in the figure above, that ontology is "Persons"), the user selects an instance from the submenu's list and then clicks the "Export Report" button highlighted in the figure to start the process of exporting a report.

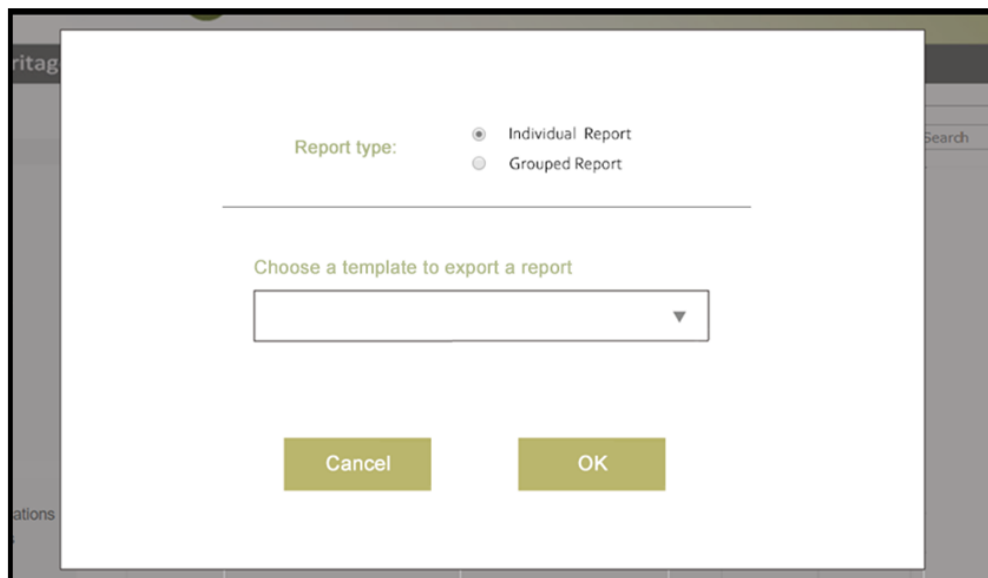


Figure 22: Mockup depicting the report export form

After clicking the "Report Export" button, the user is presented with the report export form, in which he selects the report type and template they desire from the dropdown. Finally, they must click "OK" to continue with the report export process.

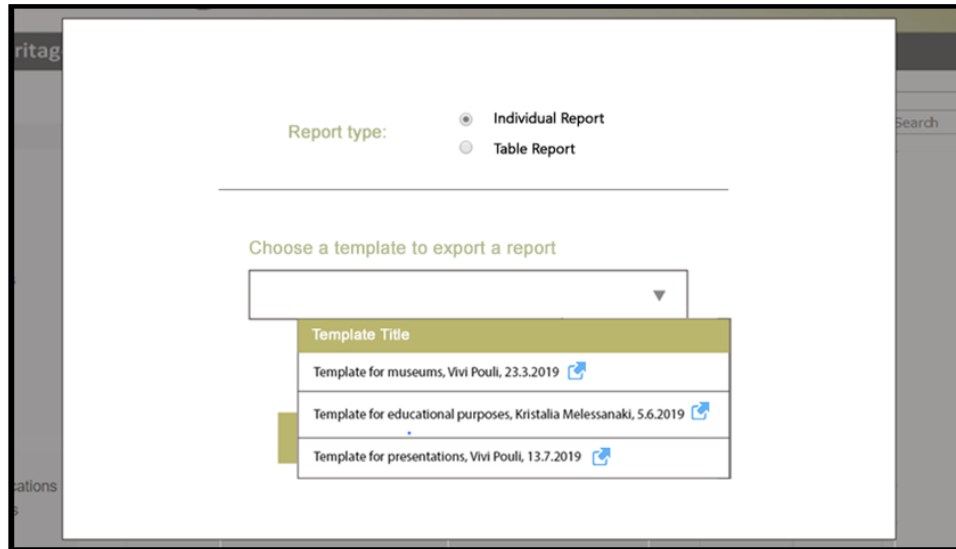


Figure 23: Mockup depicting the dropdown list of templates, populated

The populated dropdown list contains all templates compatible with the currently selected ontology. The list presents some helpful information, alongside a link to a preview (presented in this figure as a link button) which the user can click to be redirected to the template's preview.

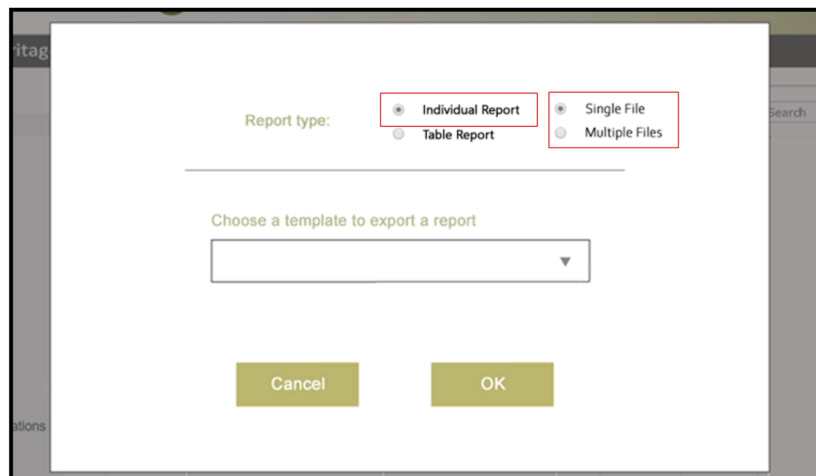


Figure 24: Mockup depicting the file grouping selection for multiple exports

As shown in the above figure, the highlighted selection on the right is only visible when the user has selected more than one instances from the ontology's submenu, indicating they desire to export multiple files in a grouped manner. This extra selection solves the issue of grouping by having the user select their preferable way of grouping. A table report will simply output all the data in a table form (in this case an XSLX file), a "Single File, Individual" report will simply merge all the instance data into one file, and finally a "Multiple Files, Individual" report will include all instances, each in their own individual report file, packaged together in an archive (in this case, a ZIP file).

6. Implementation

The program of this project is implemented based on Java for the back-end and the typical web languages for the front end, specifically HTML, JavaScript, and CSS. The back-end provides the web application with data from the database while also allowing the web application to update and expand the database. It also provides the PDF-generating mechanism. The front-end part for the program is responsible for the UI and UX, providing the necessary interaction with the user as laid out in the Design phase.

The front-end is composed of two different modes, one for managing templates and one for providing the finalized reports by previewing them and allowing them to be downloaded. The former is RTC's "Edit Mode" while the latter is the latter is its "Export Mode".

6.1. Software Libraries

- *Openhtmltopdf* is used in this program for generating PDF files using HTML and CSS. The program's front-end creates a complete HTML file for the generated report, combines it with the according CSS file and sends the HTML to the Openhtmltopdf servlet, where it gets converted into a PDF file. Following that, the PDF is sent back to the front-end where it gets downloaded.

- *Materialize* is used in this program for providing the Material Design look and feel for the front-end component of the program. Most of the components are either Materialize components or customized versions of them.

- *LZMA-JS* is used in this program for compressing and decompressing the report template data. When saving a report template file from the front-end, the program compresses the created template using LZMA-JS before sending it to the back-end. Similarly, when a report template file is loaded, provided it has been compressed, the program decompresses it before accessing it.

- *Html-docx-js* is used in this program for creating the DOCX downloadable files of the generated reports. The program's front-end creates a complete HTML file for the generated report, and directly converts it to a DOCX file using Html-docx-js without the need of a back-end implementation. After the conversion has been completed, the DOCX file is downloaded.

- *ExcelJS* is used in this program for creating the XLSX downloadable files of the generated reports. The program's front end creates an XLSX file from scratch, using ExcelJS, based on the final report's data (there is no conversion taking place) and the file is immediately downloaded.

- *JSZip* is used in this program for packing multiple generated reports into downloadable ZIP files. The program's front end first gathers all the files that need to be downloaded, each individually. Following that, the files are packed using JSZip into a single ZIP file and the ZIP file is instantly downloaded.

6.2. Interaction with the system

The creation and manipulation of the report templates is done on the RTC component of the program, using which the user can load, edit, and update templates. Since the templates are treated as entities of the SYNTHESIS system, creating and selecting templates, as well as exporting reports using existing templates are actions managed by SYNTHESIS itself. This program is being used as a plugin-in for SYNTHESIS and is called from the latter after the user has requested a relevant action. The implementation of the user's interaction with the system is carried out exactly as described in subsection 7.3 of the design process.

6.2.1. SYNTHESIS System Interaction

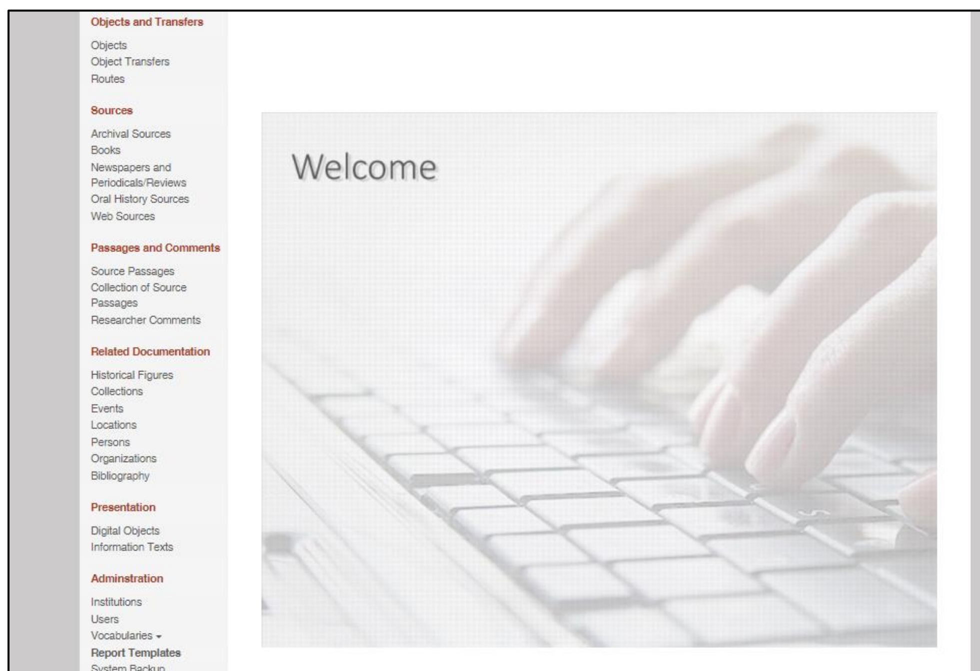


Figure 25: The SYNTHESIS index page

Initially, having logged in to the system, the administrator can see a list of all the available templates by selecting the "Report Templates" menu in the "Administration" section. The redirected page allows the administrator to select any of the existing templates for either deletion or editing. There is also the ability to create a new template, based on any of the existing ontologies from the database.

If the administrator chooses to create a new template, a new page will be loaded, where they are asked to select one of the ontologies from the system's database. Alongside that, the administrator is asked to select a template type and add the appropriate complimentary information about the template. After doing so, and submitting the form, the system creates a new Report Template ontology instance, which includes the default template file for the selected ontology. Following that, the administrator is redirected to the "Report Templates" page they were previously on where the created ontology can now be found in the list of created templates.

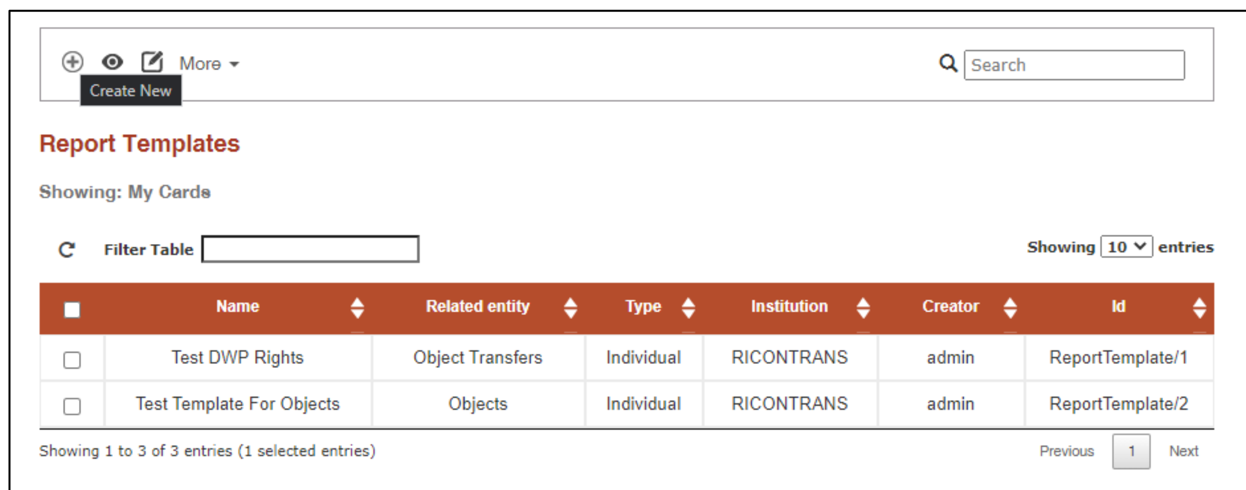


Figure 27: The "Report Templates" submenu in SYNTHESIS

The finalized layout of the "Report Templates" submenu is almost identical to the design mockups. The "Create New" button, indicated by the "+" symbol, is highlighted in this figure. All available templates are also presented in the list below the buttons.

Report Templates - Create New

Report Template Title: Test Person Template

Description: A Person Template for testing purposes

Select an entity to create a report template: Persons

Report Template Type:
- Individual (The report has the format of one or more documents containing the selected fields)
- Table (The report has the format of a table (excel) containing the selected fields)

Finish

Figure 26: The report export form

In the template creation form the user is asked to enter a template title, a description, select the ontology for the template to be based on and choose the type of the template between table and individual. The ontology selection dropdown contains all the available ontologies that are currently stored in the SYNTHESIS database. After filling the form with the required data, the user must click on the "Finish" button to confirm their selection. This final layout is again, very similar to the mockups both in terms of design and functionality.

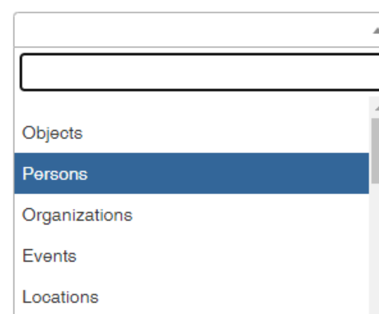


Figure 28: The ontology dropdown

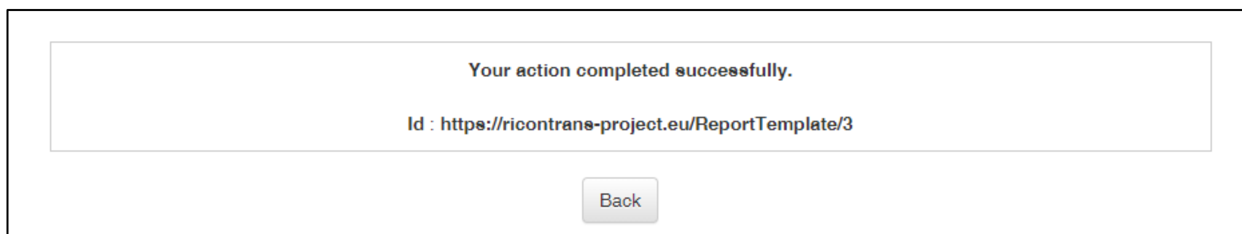


Figure 29: The template creation's confirmation message

After clicking "Finish" on the template creation form, the user is presented with a confirmation message, informing them of the successful creation of the new template and the new template's assigned Id.

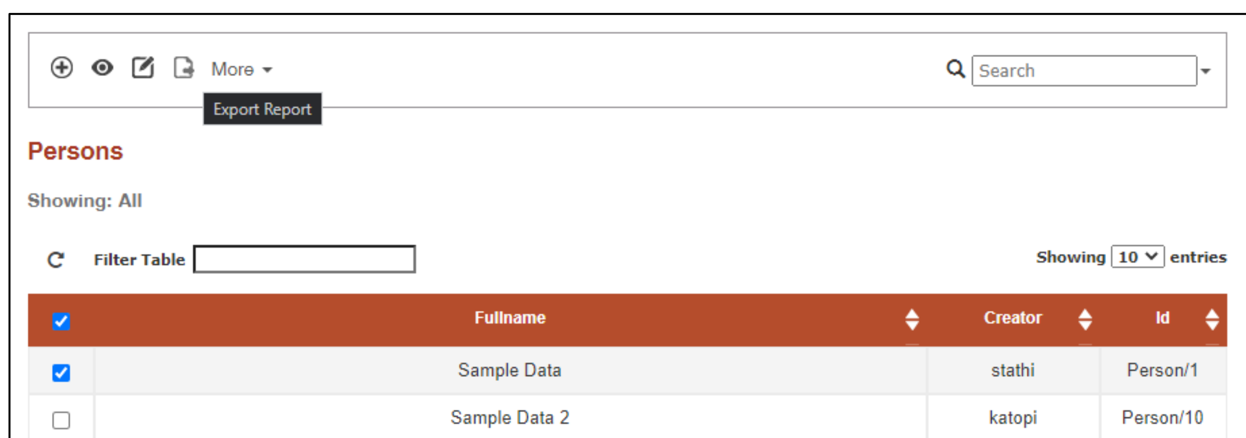


Figure 30: Selecting a template for editing in the "Report Templates" submenu

In case the user wants to edit a template, they must select an existing template instance from the list in the "Report Templates" submenu, and then click on the edit button, highlighted in the figure above. SYNTHESIS will consequently create a new window of RTC, with the selected template loaded.

6.2.2. Template Creation & Manipulation

The report template instance creation mechanism is part of the SYNTHESIS system, as described in 6.3.1. The default templates that the system includes in the report template instances are created by the program's CreateJson mechanism, which creates a JSON report template for a given ontology based on its default structure from the database. In this process all the template fields are purposely left disabled, hence for these templates to be usable, the user must edit them and enable any of the template fields.

Manipulation or editing of a template is a process managed completely by RTC. After RTC loads a template for editing as described in 6.2.1, the user can apply changes to the template, save it or show a preview of the current template format.

All the formatting styles, as well as the header and footer images are stored in the template's JSON file. This ensures that a template JSON file is a complete standalone structure.

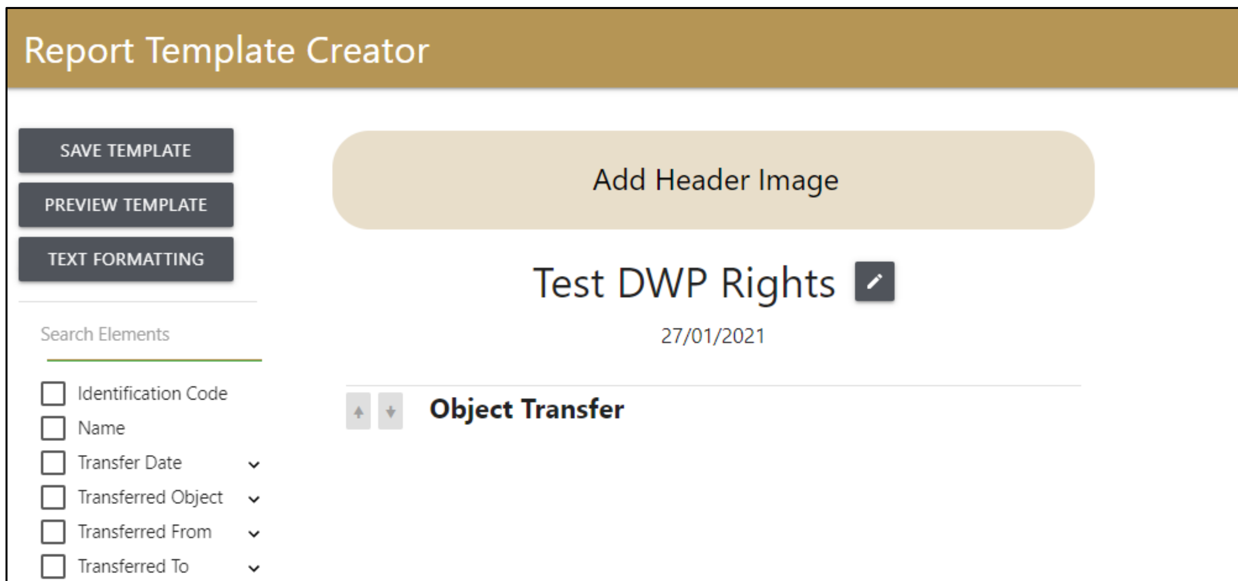
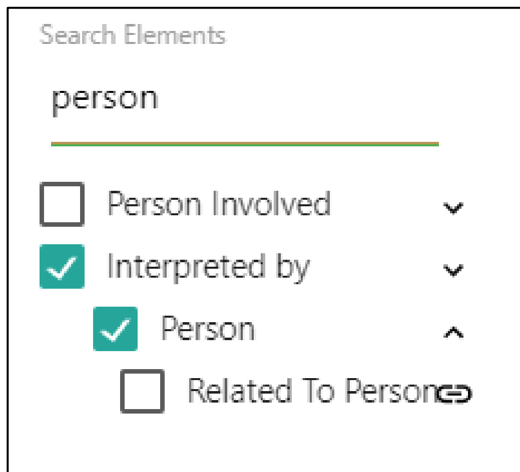


Figure 32: A newly created (empty) report template

Each template is essentially empty on creation, as shown in the figure above. This means that for any template to be usable, a user must edit them and enable at least one of the fields. This way, the template will output at least one data point. The only data that is inserted automatically is the metadata such the template title. All new template instances are based or the premade default template JSONs. This decision was not made in the initial design phase, instead it was implemented after testing the program. Initially there were specified fields pre-enabled but that quickly deemed unusable and was replaced with the final functionality.



Search functionality snippet, using element classes to filter results.

```
… // Hide items that don't match
dataTreeElements.forEach(function (elem, index) {
    let elemText =
elem.querySelector('span').textContent.toLowerCase();

    if (elemText.includes(inputString.toLowerCase())) {
        elem.classList.remove('filter-out');
    } else {
        elem.classList.add('filter-out');
    }
}); …
```

Figure 31: Search functionality

The template field tree on the left includes a search function, to quickly find a desired element. When the search yields a field that has children, its children will be visible by expanding the parent. This was also added after the initial design phase, instead it was added implemented on user feedback.

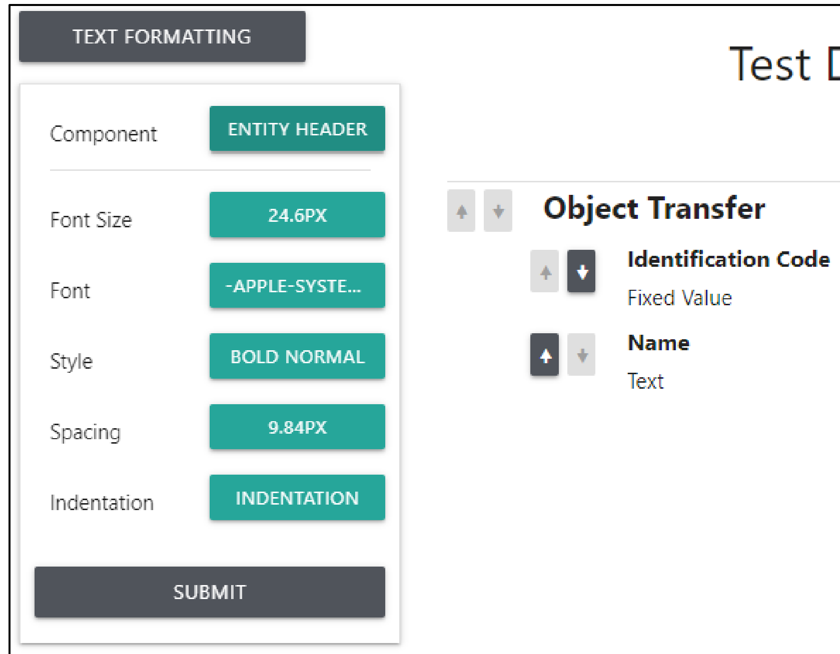


Figure 33: The text formatting menu

The text formatting menu allows for any individual element or element group to be stylized with a plethora of options. The changes are applied by clicking the “Submit” button as shown in the figure. While keeping the same basic design it had in the design phase, there were many additions, namely in the variety of options and the inclusion of individual element formatting instead of only group formatting.

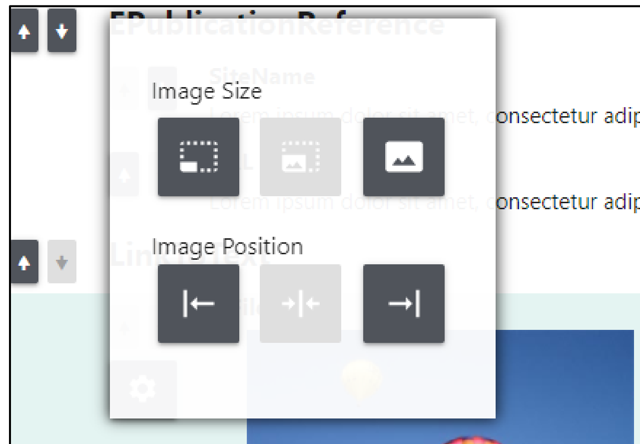
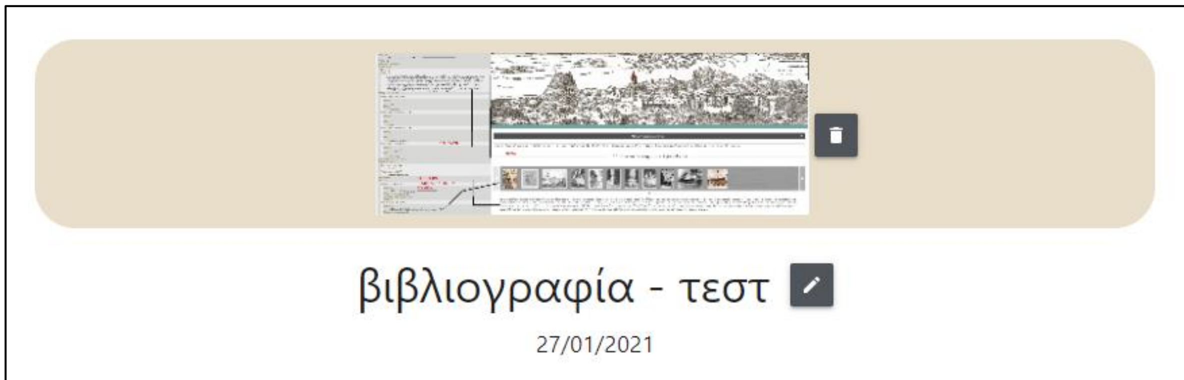


Figure 34: The image formatting menu

The image formatting menu is used on image elements to customize some image-specific properties, such as size and position. While the basic layout is kept the same from the design phase, some options like text wrapping were removed after they were proved unusable. Instead, each image element occupies a vertical space on its own.



Loading header image snippet. The image data is loaded from the Style json and then directly added to the DOM

```
…
// Set styles from presets
applyStylesFromPresets: function () {
        editableComponents.forEach(function (component, index) {...[9 lines]});

        if (stylePresets['Header'] && stylePresets['Header'].length > 0) {
                _this.addHeaderImage(stylePresets['Header']);
        }
},
…
```

Figure 35: Template header image

Users can add (and remove) their own images as headers on each template, and they are resized to fit the page accordingly. The images appear in PDF and DOCX formats but are, as expected, missing in XSLX formats as they are not needed. This is another function that was added after user feedback. The header images are stored in the JSON that represents the template itself, in order to make the template file standalone and portable. To compensate for that extra size, a compression/decompression is used when saving and loading data, respectively. The following snippet shows how this is achieved in the source code; the referenced function can also be found in Appendix 1.

```
… // Compress string with LZMA
compressString: function (inputString, compressionType) {
        var promisedReturn = new Promise(function (resolve, reject) {
                lzmaWorker.compress(inputString, compressionType, function
        on_compress_complete(result) {…[3 Lines]});
        return promisedReturn;
} …
```

Snippet 1: LZMA compression, returning a compressed string as a promise

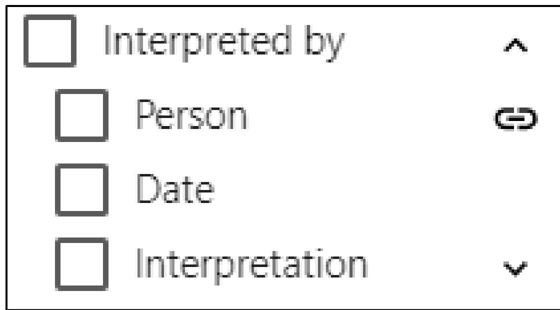


Figure 36: A linking button in the field tree

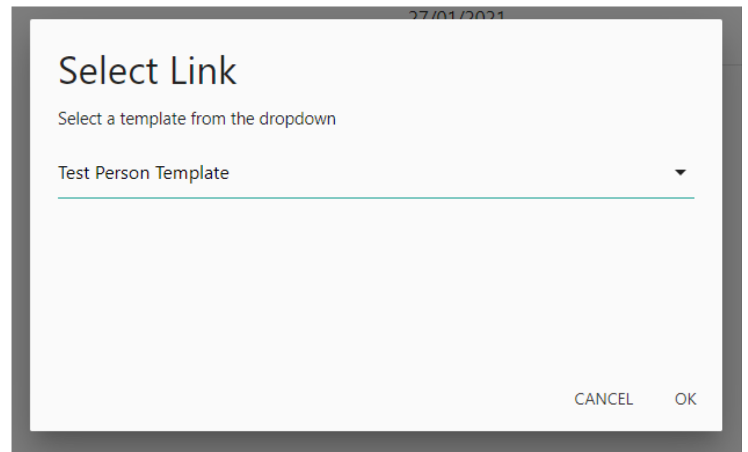


Figure 37: The linking form

Linking templates allows users to add existing templates in a new template, without having to readjust the enabled fields and order from scratch. Instead, the linked template is added as-is. First, the user must click the linking icon of the field they want to insert a link in (shown in the figure above in the “Person” element). This will cause a modal window to be displayed, where the user must select an existing template that fits the criteria of the field they are trying to link. This is done through a dropdown list in the modal window, where all the appropriate template instances can be found. After selecting the template they desire, the user must click the “OK” button and the link will be inserted. The following snippet shows the sequence of this action in the source code; the referenced function can also be found in Appendix 1.

```
… // Insert another template from link
insertTemplateFromLink: function (json) {
        …
                let jsonString = _this.convertStringToArray(xhr.responseText)[0];
                _this.decompressString(jsonString).then(function (decompressedString) {
                        …
                        var json = JSON.parse(decompressedString);
                        _this.insertTemplateDataFromJson(json, 'linked-element');
                });
        …
        };
        xhr.send(encodeURI('id=' + selectedTemplate + '&type=ReportTemplate&xpath=Data'));
}, …
```

Snippet 2: Decompressing and inserting another template as link

The linking functionality required a major redesign in RTC’s mechanisms as it was requested after user feedback, but the scope of the changes was too large to implement in a simple manner. While most post-design changes needed small adjustments, this was by far the most demanding one in terms of sensible re-design decisions.

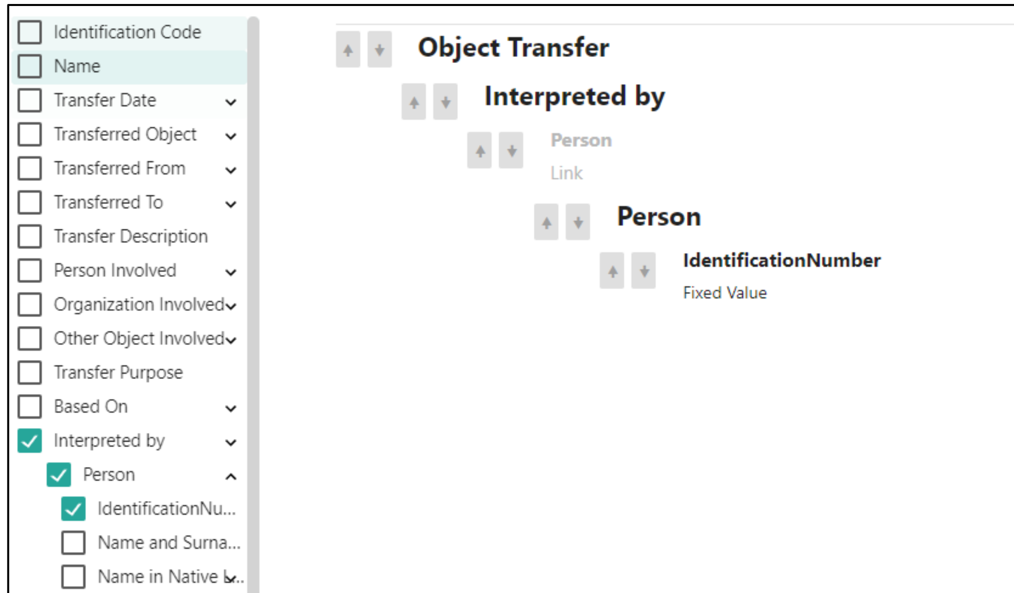


Figure 38: Example of an inserted link

Following the previous figures, the link is added in the appropriate position in the template, with all its customizations intact, bar the text formatting, which is overridden by the current template's formatting. The link field (in this case "Person" following "Interpreted by") is disabled, and hence grayed out as shown in the above figure, to avoid field duplication.

Figure 39: Preview of a customized template

Clicking the "Preview Template" button in RTC's edit mode, will display a preview of the template in its current customization. Since this functionality is part of edit mode, the data shown in the preview is sample data. Through the preview screen the user can also download the shown preview in either PDF or DOCX format. Finally, they can return to the previous screen (the main screen of RTC's edit mode) by clicking the "Exit" button, shown in the top left. This functionality follows its initial design from the design phase in functionality, but since the presentation was not solidified in that phase, some adjustments had to be made.

6.2.3. Report Generation

Generating a final report involves the merging of a report template and the one or more ontology instances' data from the database. The user must complete this action through SYNTHESIS, by selecting one or more instances for report export. After going through the template selection form, RTC is tasked with generating the final report. First, the report template file is loaded, and the report elements are inserted in the page with their appropriate formatting. Subsequently, the ontology instance data are loaded from the database and replacing the template elements appropriately, thus creating the finalized report.

The generated reports can be downloaded through a preview window provided by the program. Individual-type reports are previewed in their final form in the preview window, and the user can choose the format in which to download the report; either DOCX or PDF. Table-type reports are previewed with sample data, and the download is done automatically in the XLSX format.

Most of the report generation is done on the client-side, as determined in the initial design phase, with the only integral parts of the back-end being the PDF generation and the eXist-db query mechanisms, both of which are implemented as Java Servlets. As mentioned in the design phase, the final program's front-end uses the back-end for all calls to the database, which indeed resulted in cleaner and more efficient code for the whole project.

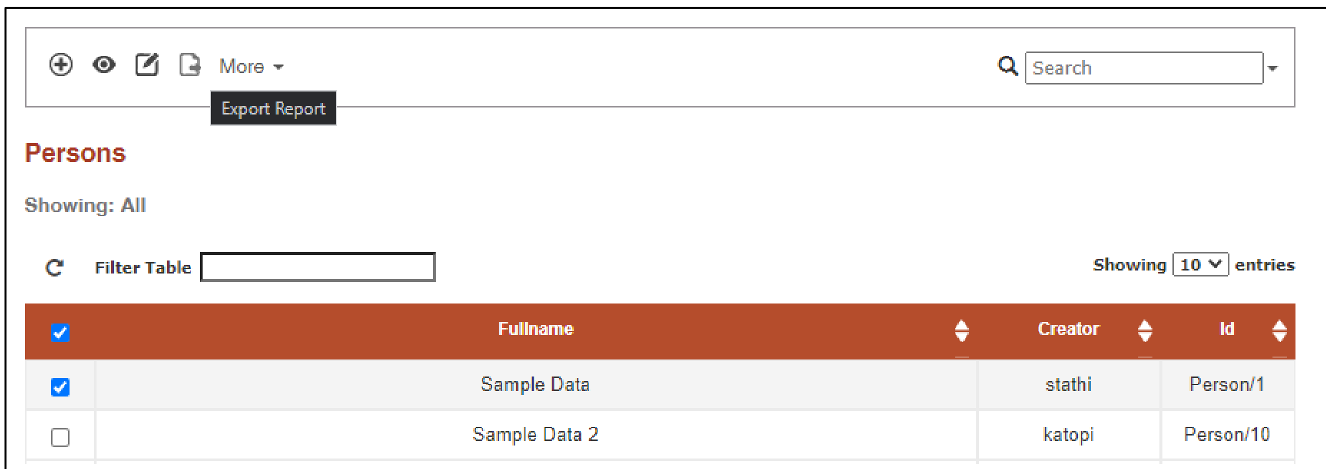


Figure 40: Selecting an ontology instance for report export

Having logged into the SYNTHESIS system, the user must click the desired ontology's button to enter the according submenu. Once they enter in the submenu, a list with all existing instances will appear, from which they must select one (or more) that they desire to export in report form. After completing the selection, they must click on the "Report Export" button as highlighted in the figure above. This will cause the report export form to appear, which will be detailed next. The report export selection process remains the same as it was in the initial design phase.

Persons - Export Report

Choose a template to export a report: Test Person Template, admin

Choose report file type for individual report: Single File / Multiple Files

Test Person Template
27/01/2021
Person
IdentificationNumber
Fixed Value
Name and Surname
Vocabulary

Finish

Overview of all the different RTC modes, including the report export form's View mode. These are essentially different views

```
…

// Set view mode UI
setExportModeUI: function () {…},

// Set view mode UI
setViewModeUI: function () {…},

// Set preview mode UI
setPreviewModeUI: function () {…},

// Set table UI
setTableUI: function () {…},

…
```

Figure 41: The report export form

The report export form contains two main selections, the desired template, and the type of report to export. The template selection is achieved using a dropdown element, similar in functionality and appearance to the ontology selection demonstrated previously. The dropdown is populated only with the templates that are compatible with the selected ontology. The report type selection determines how the reports (specifically when there are more than one selected) will be packaged or merged. A preview of the currently selected report is previewed in the lower part of the form, where RTC is loaded in view mode. View mode is similar to edit mode's preview functionality, where the template is shown using sample data, with the exception that this mode lacks all other UI elements of edit mode, as well as the file download buttons.

"Single File" reports are reports merged into one DOCX or PDF file, while "Multiple Files" reports are different individual DOCX or PDF files packaged into a single ZIP archive. Lastly, there is a template preview, with RTC loaded in view mode, to provide a quick overview of the currently selected template.

After the user has filled the form with their desired values, they must click on the "Finish" button to exit the form and move to the final export process. After doing so, RTC will load in export mode with all the appropriate data as those are selected by the user.

Figure 42: The final report, previewed in RTC in export mode

The functionality of the report form was carried over from the initial design phase with almost no changes, except that the preview is now shown in the form automatically instead of the user having to open a new window to see the preview.

After clicking the “Finish” button in the report export form, RTC is loaded in export mode, which shows a preview of the final report. In more detail, RTC in this mode first loads the template and inserts only the appropriate data (skipping the disabled fields), alongside their formatting and custom stylizations. Second, after all the template fields have been loaded, RTC uses the back-end to query the database and gather the appropriate data for each field. The final step is merging the database data with the pre-loaded template data, which results in the final report form and concludes the report generation.

Having the final report generated, RTC loads the preview and presents it to the user. This preview is an exact copy of the downloadable report, as it is this final HTML data, combined with their CSS styling that are sent to the DOCX and PDF generators, essentially converting the HTML and CSS into a document file. To download the final document files, the user must click on their desired format’s icon, colorized red with a PDF icon for PDF, and blue with a document icon for DOCX, as shown in the figure above.

In case the user selected “Multiple Reports” in the previous selection, RTC will load only the first selected ontology instance to present in its preview. When the user clicks on a format icon to download the file(s), RTC will generate all reports in the background, which is still a client-side operation, pack them in a ZIP archive and start the download of the generated ZIP file.

The final report export was not clearly specified in the initial design phase, instead it required some extensive re-designs to implement properly. After multiple iterations of the final report generation and export design, the presented process was the most satisfactory, and hence was implemented.

7. Conclusion

In an effort to increase the efficiency of analyzing results of ontology-based systems, and more specifically the SYNTHESIS system, a requirement for exporting existing data in the form of reports in different formatting styles arose, with each formatting style representing a specific report type. Through the analysis process it was clear that a templatized report generator system was needed to fulfill the task.

A templatized report engine and user interface was developed, called Report Template Creator, whose functionality would fit the nature of ontologies. Each ontology's structure is converted to a JSON object alongside all the custom formatting, which is stored as a template JSON file. Template files can be edited by users through RTC's edit mode. For exporting reports, the template file is used in combination with the ontology data from the eXist database, creating the final report(s), previewing the generated document(s) to the user, and allowing them to download the generated file(s) in their desired format, using RTC's export mode.

In conclusion, the original goal of exporting ontology data in the form of customized reports was achieved, evident by a successful testing phase. The use of templatized reporting and the creation of a templatized report engine provided the required functionality for generating reports in a workflow that fulfills the original requirements.

7.1. Future Work

The generic nature of the tool's design allows it to be used on vastly different configurations. As a result, it will be implemented as a manual page provider, where each section of the manual is a specific template field and the same template(s) will be used to present different instructions, using RTC's view mode.

More specific and customized options are already requested and will be added, such as grids consisting of images and multiple uploaded files (a SYNTHESIS feature) support. While SYNTHESIS support is built in, implementing RTC in a different system would be a straightforward task. The tool's design takes into consideration the fact that certain changes would be required in order to function in a completely different system and these changes are purposely simple to be implemented.

Bibliography

[1] D. Angelakis, C. Bekiari, M. Doerr and F. Kragianni, "Building Comprehensive Management Systems for Cultural – Historical Information," in *Proceedings of the 42nd Annual Conference on Computer Applications and Quantitative Methods in Archaeology*, Rethymno, Crete, Greece, 2015.

[2] iText, "iText: The Leading PDF Platform For Developers," iText Group nv, Inc., 2021. [Online]. Available: itextpdf.com/en. [Accessed 25 1 2021].

[3] iText, "iText DITO®," iText Group nv, Inc, 2021. [Online]. Available: itextpdf.com/en/products/itext-dito. [Accessed 25 1 2021].

[4] J. Moxter and E. Njeru, "Dynamicreports/Dynamicreports: Java reporting library for creating dynamic report designs at runtime," 9 10 2020. [Online]. Available: github.com/dynamicreports/dynamicreports. [Accessed 25 1 2021].

[5] J. Hall, "MrRio/jsPDF: Client-side JavaScript PDF generation for everyone.," 1 25 2021. [Online]. Available: github.com/MrRio/jsPDF. [Accessed 1 25 2021].

[6] E. Koopmans, "eKoopmans/html2pdf: Client-side HTML-to-PDF rendering using pure JS," 19 2 2020. [Online]. Available: github.com/eKoopmans/html2pdf.js. [Accessed 25 1 2021].

[7] N. v. Hertzen, "Niklasvh/html2canvas: Screenshots with JavaScript," 29 12 2020. [Online]. Available: github.com/niklasvh/html2canvas. [Accessed 25 1 2021].

[8] Danfickle, "Danfickle/Openhtmltopdf: An HTML to PDF library for the JVM.," 20 1 2021. [Online]. Available: github.com/danfickle/openhtmltopdf. [Accessed 25 1 2021].

[9] Science Buddies, "The Engineering Design Process," 2021. [Online]. Available: https://www.sciencebuddies.org/science-fair-projects/engineering-design-process/engineering-design-process-steps. [Accessed 25 1 2021].

[10] The Apache Software Foundation, "Maven - Welcome to Apache Maven," The Apache Software Foundation, 2021. [Online]. Available: maven.apache.org/. [Accessed 25 1 2021].

[11] A. Retter, L. Windauer, T. Krebs, J. Turner and W. Meier, "eXist-db - The Open Source Native XML Database," eXist Solutions, 2018. [Online]. Available: exist-db.org. [Accessed 25 1 2021].

[12] G. Baltusevicius, "What Is a Report Generator," Whatagraph B.V. ©, 22 5 2020. [Online]. Available: whatagraph.com/blog/articles/report-generator. [Accessed 25 1 2021].

[13] R. J. Burke and B. A. Reinhart, "Templatized Reporting Engine". Texas, USA Patent US20130054284A1, 26 11 2011.

[14] Google, "Google Docs: Free Online Documents for Personal Use," Google LLC, 2021. [Online]. Available: www.google.com/docs/about/. [Accessed 25 1 2021].

[15] WordPress, "WordPress Editor," WordPress, 17 8 2020. [Online]. Available: wordpress.org/support/article/wordpress-editor/. [Accessed 25 1 2021].

[16] W3Schools, "XML Introduction," W3Schools, 2021. [Online]. Available: www.w3schools.com/xml/xml_whatis.asp. [Accessed 25 1 2021].

[17] Materialize, "Materialize: A modern responsive front-end framework based on Material Design," 2021. [Online]. Available: materializecss.com. [Accessed 25 1 2021].

[18] A. Wang, "Dogfalo/materialize: Materialize, a CSS Framework based on Material Design," 1 6 2020. [Online]. Available: github.com/Dogfalo/materialize. [Accessed 25 1 2021].

[19] Evidence Prime, "EvidencePrime/Html-Docx-Js: Converts HTML documents to DOCX in the browser," Evidence Prime, 17 5 2016. [Online]. Available: github.com/evidenceprime/html-docx-js. [Accessed 25 1 2021].

[20] A. Wang, A. Lubbe and S. Pawel, "ExcelJS/ExcelJS: Excel Workbook Manager," 30 11 2020. [Online]. Available: github.com/exceljs/exceljs. [Accessed 25 1 2021].

[21] S. Knightley, "Stuk/JSZip: Create, read and edit .zip files with Javascript," 16 1 2021. [Online]. Available: github.com/Stuk/jszip. [Accessed 25 1 2021].

[22] LZMA-JS, "LZMA-JS/LZMA-JS: A JavaScript implementation of the Lempel-Ziv-Markov (LZMA) chain compression algorithm," 19 11 2017. [Online]. Available: github.com/LZMA-JS/LZMA-JS. [Accessed 25 1 2021].

[23] The European Commission, "EUROPEAN UNION PUBLIC LICENCE v. 1.2," 18 5 2017. [Online]. Available: https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/eupl_v1.2_en.pdf. [Accessed 25 1 2021].

[24] Joinup Europa, "Matrix of EUPL compatible open source licences," Joinup Europa, 2021. [Online]. Available: https://joinup.ec.europa.eu/collection/eupl/matrix-eupl-compatible-open-source-licences. [Accessed 25 1 2021].

[25] Free Software Foundation, Inc, "GNU Affero General Public License," Free Software Foundation, Inc, 19 11 2007. [Online]. Available: www.gnu.org/licenses/agpl-3.0.en.html. [Accessed 25 1 2021].

Appendix

1: Code

ReportTemplateCreator.js outline

```
// @author Manos Paterakis
var reportTemplateCreator = {

…

// Initialize components
init: async function () {},

// Set language strings
setLanguageStrings: async function (lang) {},

// Initialize data tree search funcitons
initDataTreeSearch: function () {},

// Filter Search Tree
filterSearchTree: function (inputString) {},

// Adjust Sidebar Height
adjustSidebarHeight: function () {},

// Set report date to today
setDateToToday: function () {},

// Set template properties
setTemplateProperties: function () {},

// Edit template name
editTemplateTitle: function () {},

// Set new template name
setTemplateTitle: function (title) {},

// Load template metadata
loadMetadata: function () {},

// Load templates
loadTemplates: function () {},
```

```
// Adjust template select
adjustTemplateSelect: function () {},


// Remove hidden elements from JSON
removeHiddenElementsFromJson: function (json) {},

// Insert another template from link
insertTemplateFromLink: function (json) {},

// Insert another json from link
insertTemplateDataFromJson: function (json, _optCustomClass) {},

// Remove all template link
removeTemplateLinks: function () {},

// Remove a template link
removeTemplateLink: function (link) {},

// Check whether json is empty
checkIfEmpty: function (json) {},

// Set view mode UI
setExportModeUI: function () {},

// Set view mode UI
setViewModeUI: function () {},

// Set preview mode UI
setPreviewModeUI: function () {},

// Set table UI
setTableUI: function () {},

// Check and hide settings sections
checkAndHideSettingsSections: function () {},

// Add a header image
// https://stackoverflow.com/a/40971885
addHeaderImage: function (_optImageData) {},

// Remove header image
removeHeaderImage() {},
```

```
// Export helper function
exportHelperFunction: function () {},

// Preview template
previewTemplate: function () {},

// Exit preview mode
exitPreview: function () {},

// Create a table preview
createTablePreview: function (excel) {},

// Generate standalone HTML from current page
generateHTML: async function (exportFileType, optInputHTML) {},

// Automatic pagination
automaticPagination: function (html) {},

// Load db data on current page
loadDbData: function (optInputHTML, optId) {},

// Clone children elements
cloneChildrenElements: function (parentElem, idValue, idIndex, _optPrevIdValue) {},

// Create file URL
getFileURL: function (filename, id, type) {},

// Download sample file
downloadSample: function () {},

// Create PDF file
createPDF: async function () {},

// Create docx file
createDocx: function () {},

// Create excel file
createExcel: function (_returnFileOpt, _sampleDataOpt) {},

// Convert string array to string[] object
convertStringToArray: function (inputString) {},

// Save template
```

```
saveTemplate: function () {},

// Create JSON file from current elements
createJsonFromElements: function (elementOpt) {},

// Get children json
createChildrenJson: function (parent) {},

// Set styles from presets
applyStylesFromPresets: function () {},

// Set style presets
setStylePresets: function (presets) {},

// Get Style json component
createJsonStyle: function (_optSelectedElement) {},

// Get element's style in a single string
getElementStyleString: function (element) {},

// Get inner elements' style in an array
getInnerElementsStyleArray: function (element) {},

// Force close inputs
forceCloseInputs: function () {},

// Indent children elements
indentChildrenElements: function (_optParent, _optIndent) {},

// Unindent children elements
unindentChildrenElements: function () {},

// Show children paths
showChildrenPaths: function () {},

// Hide children paths
hideChildrenPaths: function () {},

// Apply style to selected element
setElementStyle: function () {},

// Create an element
createElement: function (elementType, elementTitle, elementValue, indentOpt,
parentOpt) {},
```

```
// Load style for a specific element from JSON
loadElementStyle: function (element, styleJson) {},

// Force important styling on elements
forceImportantStyle: function (elem) {},

// Hide parents with no children
hideEmptyParents: function () {},

// Get element name from title object
getNameFromTitle: function (title) {},

// Get element entity from title object
getEntityFromTitle: function (title) {},

// Get page break from title object
getPageBreakFromTitle: function (title) {},

// Get exclusion from title object
getExclusionFromTitle: function (title) {},

// Get style from title object
getStyleFromTitle: function (title) {},

// Create page-break button
createPagebreakButton: function (parentRow) {},

// Toggle pagebreak
togglePageBreak: function (element) {},

// Create element customization button
createCustomizationButton: function () {},

// Set a custom item for formatting
setCustomElementFormatting: function (element) {},

// Handle custom element switch
adjustCustomElementSwitch: function () {},

// Create table elements
createTableElement: function (elementType, elementName, elementValue) {},

// Create label-changing HTML
```

```
createLabelChangingHTML: function (elementName, htmlName, row,
_opt_buttonsArray) {},

// Crate a toolbox
createToolbox: function (indent, elementType) {},

// Adjust buttons inside toolbox
adjustToolbox: function (element) {},

// Adjust image settings toolbox
adjustImageToolbox: function (element) {},

// Adjust list settings toolbox
adjustListToolbox: function (element) {},

// Move element up
moveUp: function (element) {},

// Move element down
moveDown: function (element) {},

// Open image formatting settings
openImageFormatting: function (element, buttonFormatting) {},

// Open list formatting settings
openListFormatting: function (element, listFormatting) {},

// Merge image with other element
mergeWithImage: function (element) {},

// Align image
alignImage: function (element, option) {},

// Set image size
setImageSize: function (element, option) {},

// Set list type
setListType: function (element, option) {},

// Align list text
alignList: function (element, option) {},

// Separate image from other element
separateFromImage: function (element) {},
```

```
// Swap elements with visual transition
swapWithTransition: function (element1, element2, overrideTransition) {},


// Show template properties edit menu
showTemplatePropertiesEditMenu: function () {},

// Toggle text formatting tools
toggleTextFormattingTools: function (_optSelectedItem, _optForceSelectedItem) {},

// Disable required formating toolbox buttons
disableRequiredButtons: function () {},

// Enable required formating toolbox buttons
enableRequiredButtons: function (selectedElem) {},

// Update components dropdown list
updateComponentsDropdown: function (_optSelectedItem, _optForceSelectedItem) {},

// Fade out an element
fadeOut: function (element, callback) {},

// Fade in an element
fadeIn: function (element, callback) {},

// Create elements from JSON
createElementsFromJson: function (json, indentOpt, parentOpt) {},

// Add listeners on children of element
addChildrenListeners: function (element) {},

// Highlight element
highlightElement: function (element, color, transitionDuration, borderRadius) {},

// Stop highlighting element
stopHighlightingElement: function (element) {},

// Show element and child elements
showTreeElements: function (element) {},

// Hide element
hideTreeElements: function (element) {},
```

```
// Show element
showTreeElement: function (element) {},

// Hide element and child elements
hideTreeElement: function (element) {},

// Show element and child elements
enableTreeElements: function (element) {},

// Hide element and child elements
disableTreeElements: function (element) {},

// Toggle tree element visibility
toggleTreeBranchVisibility: function (element) {},

// Toggle tree element state
toggleTreeBranchState: function (element) {},

// Get tree element name
getTreeElementName: function (element) {},

// Show element in preview
showElement: function (element) {},

// Hide element in preview
hideElement: function (element) {},

// Create data tree from given data
createDataTree: function (data, depth, parent, isArrayData) {},

// Select teamplate for link
selectLink: function (element) {},

// Update tree view
updateDataTree: function () {},

// Collapse empty parents
collapseEmptyTreeParents: function () {},

// https://stackoverflow.com/a/722732
// Traverse JSON
traverseJson: function (jsonObj, func, depth, parentOpt) {},

// Get next element of same indent
```

```
getNextOfSameIndent: function (element) {},

// Get previous element of same indent
getPreviousOfSameIndent: function (element) {},

// Get children
getChildren: function (element) {},

// Get direct children
getDirectChildren: function (element) {},

// Get children in sequential order
getChildrenSequential: function (element) {},

// https://stackoverflow.com/a/901144
// Get url params
getParameterByName: function (name, url) {},

// Convert image data to base64
convertImagesToBase64: function (html) {},

// Compress string with LZMA
compressString: function (inputString, compressionType) {},

// Decompress string with LZMA
decompressString: function (inputString) {},

// Parse array string and return an Array object
parseArray: function (input) {},

// TESTING
loadTestData: function () {}

…

};
```