Security Assessment Automated Reporting

Diogo Torres
diogo.torres@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2020

Abstract

A Report provides a way to present information in an organized format for a target audience and purpose. A summary of a report can be delivered orally, but complete reports are mostly transmitted in the form of written documents[6]. Most businesses, especially in IT, must showcase their work and performance through the creation of reports. Yet this is a task that requires a large amount of time and effort. The aim of this thesis is to capture the repetitiveness and predictability of report production, transforming it into a partially automated task. This leads to a decrease in user time, labor and errors, and a new way of creating reports. We intend to deliver a platform for the users to manage and distribute the reports amongst team members. In this work, the challenge of report creation is tackled by reducing a report as a whole into both static and dynamic building blocks. This means the user will spend less time rewriting repetitive information and focus his efforts into writing the case-by-case data relevant to each report. The management of reports will be achieved by giving the users a table containing the reports relevant to them and enabling them to create, delete and modify those documents. The result of this thesis will be a reporting module composed of three main components. First, a server responsible for manipulating: users, clients and the produced reports inside the connected databases. It also provides an API for the client application to make calls to functions implemented in the server. Second, a client application that gets the necessary information from the databases using the server-side created API, sending data to be stored, using the same API. Finally, an interface that presents a user login system, a report repository and, most importantly, a report editor.
Keywords: Report, Project, Issue, Knowledge Base, Service, Repository, Editor, API, User

1. Introduction

Communication is a key characteristic of a successful relationship between a company and its costumers. This is especially true in software engineering, where the product usually takes the form of source code. However the end user is only interested in the final computer program and how to use it.

This is where a healthy flow of information makes the transaction worthwhile. Proper project documentation of the work aims to inform the client of what was done, how it was done and how valuable it is to the client. This documentation is also important to the producing company as a way of archiving its endeavors for future reference.

A report is a document created to deliver a specific set of information requirements to a certain group of people [20]. As such, it presents itself as a possible way to record the service provided and it is, in fact, a perpetual reality of any IT company. Reports are a way of reducing great amounts of data into the fundamental knowledge points, making it possible to rapidly acquire high level knowledge without needing to be aware of all the details of the day-by-day development process.

The type of information gained by creating reports lets both client and team reach a common ground of understanding about the created product so that both entities are able then to make critical and knowledgeable decisions for the steps to take place in the future.

Good quality reports, like any other type of work, are generated not by one person, but with team effort. For it to be possible, an infrastructure for team members to orderly interact has to be in place. This infrastructure comes in the form of management tools for users, clients, projects and a framework capable of storing all this data; the creation of this infrastructure is the core of our work.

1.1. Motivation

The subject of this thesis was introduced by MAINSEC and came to fruition based on challenges they face on a daily basis.

MAINSEC is an IT company with key expertise in information security and IT security consulting, management and professional services. The team at

MAINSEC engages with other businesses to inspect and report the security state of their applications and infrastructure.

Right now, MAINSEC is presented with the challenge of optimizing the process of composing reports, due to the repetitive nature of this task. This type of work usually results in unnecessary strain for the writers and inevitably causes mistakes in production which lead to more wasted time and effort.

After creating the reports there is a need to share files between team members and clients. This is accomplished with the use of basic file managers over online file hosting services and email correspondence.

With regards to archiving and administration of files handled by the team, there is a problem with control, privacy and security, or the lack thereof, over the files stored in external services like Dropbox or Google Drive. The projects themselves also lack an official and updated log of the many states that projects traverse.

This lack of structure brings about wasted time and a lack of control. It causes team members to not know what or when to do their respective tasks.

1.2. Contributions

Our contributions and, in particular, the contributions of this thesis are the following:

1. Provide a notion of what report creation is, why it is relevant and what are its current faults;

2. Study existing solutions and related work to target this problem;

3. Detail the specific requirements requested by MAINSEC, regarding the task at hand;

4. Propose and design a solution for this problem, comprised of: a User dependant Login system that changes the way the application behaves relative to the permissions given to that User; a Projects and Reports Repository for team members to manage attached team members and Clients, change meta information of these files (deadline, status,etc) and provide a way to easily find and open Reports for editing; and a Report Editor that encapsulates both the capabilities of a common text editor and the custom functionalities that enable the rapid creation of these specific Reports.

5. An evaluation of the created solution by way of a comparison with a similar system that is already in place.

2. Background

In the following sections we will present some tools and technologies used for report creation, editing and storing that served as inspiration for the development of our own software.

Report Creation Tools

2.1. SAP Crystal Reports

SAP Crystal Reports is a system designed to take vast existing databases and present them as ready-to-consume information in the form of reports that people, both internally and externally, can use to keep informed and make better decisions. To reach this goal SAP Crystal Reports gives the user an editing software to create pixel-perfect reports that reflect great amounts of information as concise and easy to consume documents.

By creating reports using formulas, cross-tabs, sub-reports and conditional formatting, the user can get a clear picture of a hard to understand large data set as well as uncover conclusions that would otherwise be hidden.

This tool enables the user to connect to almost any source of data like large CSV files or NoSQL databases to then produce and distribute them as one of many popular formats like PDF, Word or HTML [23, 22].

2.2. TIBCO Jaspersoft Studio

Much like SAP Crystal Reports mentioned before, this editing software was made to design and run report templates using a plethora of visual components like charts, tables and maps.

Using Jaspersoft Studio, one can access different types of data sources, including big data, CSV, Hibernate, Jaspersoft Domain, JavaBeans, JDBC, JSON, NoSQL, TIBCO Spotfire® Information Links, XML, or your own custom data source[25]. After creating the layout of the report, the user is capable of exporting in many popular formats that fit any data need, like PDF and spreadsheets or raw CSV and XML documents.

2.3. Overleaf

Overleaf is a cloud-based LaTeX editor that allows multiple users to write, edit and publish scientific documents [16, 13]. Overleaf provides the convenience of an easy-to-use LaTeX editor with real-time collaboration and the fully compiled output produced automatically in the background as you type.

Having to write in the form of source code requires a previous knowledge of the LaTeX language to produce documents, but creating a document with LaTeX frees the user from worry around design as this system provides the user with plenty of industry standard layouts and formatting [10].

Overleaf lets the user create templates of documents to be completed for each case, as well as publishing to mainstream supported file formats.

File Management

2.4. Google Drive

Google Drive is a cloud-based multi-platform file storage and synchronization service [4].

The service is used by individuals and teams to store, manage and share files online. Users can choose who they share their files with and what permissions they have on those items. They can share access to folders or individual files.

As this tool is a part of the Google Office Suite, from Google Drive a user can open a document for collaborative editing in the other office applications, like Google Docs.

2.5. File Browser

The generic file browser is the most basic way of storing and managing files. It is included in every main Operating System and is used to store files offline and manage these files in a folder structure. If a user intends to share it with a team member, it must be done using external tools. At most, one can set read/write permissions per file or folder.

Database Systems

2.6. PostgreSQL

PostgreSQL is an open source object-relational database that aims to provide access to large data sets even in the heaviest workloads.

PostgreSQL is highly extensible, letting the developer define his own data types, custom functions and write code from different programming languages without the need to recompile the database.

The SQL standard is mostly respected by PostgreSQL, with a few exceptions where this compliance would hinder features or architectural structure. Most features and functions that form the SQL standard are supported even if sometimes with a different syntax [24].

2.7. MongoDB

MongoDB is a document oriented database with the scalability and flexibility that the user wants, whilst offering the querying and indexing needed.

By storing JSON-like data documents, MongoDB enables each document to have its own fields and data structure and be able to change these over time [21].

Web API

2.8. GraphQL vs REST

GraphQL is a query language for APIs, as well as a run-time for fulfilling queries related to the user's data. Unlike other REST APIs, GraphQL provides clients with the power to request just what they need and nothing more through the implementation of a schema. Apart from being able to use a schema, the user is also able to change that schema over time and evolve the APIs or with the APIs with which it communicates.

Apps using GraphQL are fast and stable because they control the data they get, not the server. This means that by defining the requested data in a given query, the user only receives that data in question and not the unfiltered content provided by a server.

With GraphQL the user can write APIs that leverage existing data and code with GraphQL engines available in many languages. This way, the app only has to be concerned with a single API to access multiple and varied storage engines [17].

Web Application Framework

2.9. Angular

Angular is a platform made for development of web applications. Much like its former version, as AngularJS, Angular remains a popular option for its purpose. Its popularity comes from its capabilities regarding concepts like data binding, which makes dynamic page updates possible, directives that enable the users to create their own HTML tags and therefore a more personalized application. Furthermore, there is dependency injection to easily create reusable and testable code.

Angular is a fully re-written and redesigned version of AngularJS and now is the one that has become most widely used and actively maintained. Angular builds upon its predecessor by using TypeScript which is similar to JavaScript, so the user does not have to learn a whole new language, and extends the capabilities of AngularJS by being web, mobile, and native desktop among many other improvements [18, 19].

2.10. Vue.js

Vue.js is an open-source model–view–viewmodel progressive JavaScript framework for building user interfaces and single-page applications [9, 11]. Vue offers much of the same capabilities of Angular, but it does not require the use of Typescript and offers a much less opinionated perspective. This means that a user has a smaller learning curve, if he already knows how use vanilla Javascript, and has more flexibility on how to develop an application as Vue doesn't impose a Right Way to build an application [5].

Discussion

In this chapter, we presented relevant systems in the area of Report Creation and what other tools can be used to achieve a complete team framework for managing and storing reports. Yet, no tool is without its imperfections, so we highlight the major faults and features of each type of tool to learn the most from them.

From the Report Creation applications we want to replicate the capability of using a template document, to be filled by the end user and to export as a PDF file, allowing any Client to see it or read it. From Crystal Reports and Jaspersoft Studio, we want to reproduce the capacity for automatically getting data from multiple sources into our doc-

ument, but we want to use open-source software. Like Overleaf, we must implement an application capable of being used online by multiple users and to have the document automatically saved to the cloud, but we want to control the server where the documents are saved to and for the text editor to behave differently according to the user in session. What is missing from these tools is the custom functionality of adding Issues (Section 3.2), specifically, or even the capability to add other modular functionality as needed.

Google Drive, like any file browser, presents the files and folders in an ordered manner, but lacks specific information about said files that need to be present for the user to correctly manage them. Once again we need a system that acts in a particular manner depending on the user, beyond the read/write permissions provided by booth Google Drive and a file browser. The possibility to go from the file management software to the editing software is to be kept. The entities managed by repository are more complex than folders and files, we need a way to represent Clients, Users, Projects and Reports and there is no way to produce this functionality in either of the tools presented.

The database systems we introduce above satisfy our needs as they are already being used at MAINSEC and are an compulsory development requirement (Section 3.7).

To develop the web API, we focused on the advantages of GraphQL, because we must use different databases to store, manipulate and provide data to the client application.

Angular and Vue.js present a very similar offering. Since there was already past experience with Angular, this is its main advantage.

3. Requirements

In this chapter we show the guidelines to be followed and goals to be met during development, according to MAINSEC.

Functional Requirements

Hereinafter, we will describe which requirements must be met within each application interface and which requirements belong to their respective actor. So firstly, we will define the roles of Auditor, Reviewer and Project Manager.

3.1. Entities

1. The Auditor is the person that will interact directly with the existing application provided by the client and then write the report which contains all the issues found during the assessment. This is the sole entity responsible for creating new reports and directly change their contents and may have to do so repeatedly with each review of the document until it is approved by the Reviewer;

2. The Reviewer is the one who reads and makes suggestions for corrections to be made in the reports created by auditors;

3. The Project Manager is responsible for the management of the ongoing projects and guarantees that these are delivered properly. The management of projects includes creation of projects and reports, dictating the roles of the team members involved in each project, and giving the final validation of a report before delivery to the client.

3.2. Main Objects

The paragraphs below describe the objects that are stored and manipulated by the actors detailed in section 3.1.

Project

The Project is principal organizational component of this solution. The Project incorporates the information about the team members assign to it and which roles they play.

Each Project has one of three statuses: “Open”, “In Progress” or “Closed”.

Projects hold the Reports the Auditors write after an assessment of a Client application. Each Project has an attached Client.

Report

A Report is the document produced to describe the issues found by the Auditor during an application assessment.

In a management perspective, the Report keeps information about its deadline for review and delivery, and the different statuses it traverses through.

A Report starts its life in a “Open” status, changes to “In Progress” as the Auditor starts writing in it, then enters a cycle. This cycle begins when the Auditor finishes the first draft of the document and sets the Reports to “Review” and waits for the Reviewer actor to read and append comments as needed.

When the Reviewer finishes reading the produced content and writing his suggestions to the Auditor, he sets the Report to a “Reviewed” state.

The third phase of the review cycle is entered when the Auditor notices the newly written suggestions of the Reviewer and starts to edit the Report accordingly, setting the status to “In Progress”.

Th final status is “Closed”, which is set by Project Manager when he has read the Report and is confident to deliver it to the Client.

Issue

Issues are the main components that distinguish Reports. After the Auditor has finished the assessment on the application provided by the Client, the Auditor uses the Report to showcase every fault found.

The Auditor is tasked with matching the faults, found during the Client application assessment,

with the corresponding Issue that best describes them. Having found the correct Issue, the Auditor must place the static contents of the Issue, retrieved from the Knowledge Base (3.6), in two sections of the document.

The two sections of the Report that list Issues are the "Summary of Assessment Results" and "Assessment Details". In the "Summary of Assessment Results" section, Issues are listed in a table with their title, description and what is the subsection where this Issue is described in detail, in the "Assessment Details" section. In the "Assessment Details" section, the Issue will be described in full detail, bringing from the Knowledge Base (3.6), all the static information about this Issue.

The second part of every Issue is dynamically created by the Auditor, to explain how the Issue relates to the specific fault in the Client program. This dynamic data is written inside the subsection the Issue occupies in the "Assessment Details" section of the report. This dynamic data is composed of text and images that explain the state of the fault and what the Client could do to correct it.

3.3. Report Repository

The Report Repository offers a view of all projects and reports to be accessed and managed by the Auditor, Reviewer or Project Manager.

The interface presented to users must take the form of a table listing projects and reports. The columns delineate features like name, status, deadlines, team members, etc.

There are capabilities exclusive to each user and a few that can be executed by all of them. We will detail the actions that each entity is capable of.

All parts involved can sort and filter through all active projects and reports for further actions. For example, in a table of projects, the user can use a search bar to find a project using a keyword. To change the order the projects are displayed, a user can choose, for example, to sort them by status.

Upon selecting a Report any user can preview how it looks when finished and save it as a PDF file.

Although all entities are able to change the status of a Report, there are different statuses available to different team members.

A new Project can be created by anyone, but only the assigned Project Manager can change and delete the Project from then on. The meta information edited by the Project Manager includes the Project name, team members, target Client and status.

Report initiation and deletion is exclusive to a Project Manager and its Auditor, and during is writing both these actors are able to change its deadlines. Only the Auditor has the right to write the Report.

Lastly, the Reviewer is the entity solely responsible for opening the Report to add comments for the Auditor to read afterwards.

3.4. Report Editor

The Report Editor presents a single interface for two different users, but each user will get a different set of functionalities. These two users will be the Auditor and the Reviewer.

The layout must contain three main features: scrolling pages representing the opened document, just like any other word processor, where the Auditor sees and fills a report as a regular document in any other text editor; a sidebar with a search engine that pops up when an Auditor wants to insert a new Issue into the document, helping him, more easily, find the right Issue; and finally, a comment section for the Auditor to read and the Reviewer to write on.

The report must follow the existing ordered structure where the document is a composite of static and dynamic data. Some components are already partially constructed and present in every report and, then, the user only has to fill in the blanks. Other parts are uniquely added to each report, as is the case with the Issues.

An Issue is an object that can be inserted into the document, inside its respective section. Each of these Issues is made of three components: a static text description that is imported from the Knowledge Base (page 6); the technical details and current state to be manually filled in by the user; and the severity which determines where the *Issue* is placed in the document.

The comment section is placed alongside each respective page, i.e., a comment section only contains comments relevant to that page. The Reviewer add and modify annotations, while the Auditor only sees what the Reviewer wrote before.

Development Requirements

This section will specify how the new components will work, their functionalities and how they will interact with the existing system. Additionally, we will introduce how some existing micro-services operate, as they are crucial to the new implementations. Finally, we will describe what technologies MAINSEC requires to be used in our solution, as they are already in place.

3.5. Report Service

The Report Service will be a composition of three components, a back end server that directly connects to the required databases, the Web API definition and the a client application where the Front end is implemented.

The back end server has the responsibility of creating the necessary functions to manipulate data

on multiple types of database systems. This is also where we will implement the new objects and respective operations needed for representing the Reports and necessary infrastructure around them.

A Web API must be defined as an abstraction layer for the client application, removing the need for the client to directly interact with data sources and having the advantage of only being required to use a single language to access those different databases. This Web API is what enable the connection between the server and client application layers.

Finally, the Report Service is where we will build the Front end interface to be utilized by the team members. This interface lets a team manage, create, edit and, lastly, export the result Reports to deliver to the Client.

3.6. Related Services

The services described below represent the infrastructure in which the the service produced in this thesis must integrate to.

User Service. A micro service responsible for every aspect of user management in the platform. Registration, authentication, authorization and removal of clients and their information are all functionalities associated with this service.

Knowledge Base Service. Will provide the database to be used as a source for searching Issues and later populate the created reports. Besides hosting data relating to Issues this service will hold other types of the static data required for the production of these reports such as Disclaimers, Terms of Use or other textual descriptions relevant to the client. Data creation, deletion and modification will be done exclusively by the Knowledge Base Service.

Project Service. Similarly to the Knowledge Base Service, serves as a system for storing and maintaining data. In this case, it will manage projects themselves and all the information related to each project. Through this service a team members can create, delete, and modify projects. This includes attached reports, client information, assigned team members and scheduling.

3.7. Technologies

Our application has to be able to connect with the database systems, PostgreSQL and MongoDB. Having these databases already in place, we are required to use them to store the newly created information from Projects and Reports.

Both Angular and GraphQL were not required options, but had the great advantage of already had being use by the team members at MAINSEC that could help during development.

4. SAAR2020

4.1. Approach

Our approach started by understanding the existing services in place at MAINSEC. Looking at the services, we found what information they store and how to transfer that data between the services and our new application.

The User Service and Project Service use the PostgresSQL database system to store, respectively, user and projects information. The Knowledge Base Service is using MongoDB to store the static Issues and as referred in section 3.7, our Report Service must use this same database to store the reports as they are created.

To bridge all the required databases, existing and the ones formed during development, a better approach is a server that is capable of connecting to different databases, using distinct protocols and provide an interface to communicate with them all using only one protocol.

To satisfy the needs previously mentioned, we designed a GraphQL API using a NodeJS server that enabled the front-end to perform CRUD(create, read, update and remove) operations in our databases.

After making sure we were able to connect with the existing databases, we moved on to expanding the existing objects, Projects and Users, and created a way to archive Reports. All of this was possible using GraphQL.

Having the basic capabilities of our server working, we started building our interface in Angular. Based on what users at MAINSEC already used before and their current unmet needs, we started work on a Report Repository similar to most file storage services. This means, a table showing the Projects and/or Reports belonging to a given user, that showed in its columns some meta information (name, status, etc.) and the actions that the user could perform.

Regarding the Report Editor, we opted for a WYSIWYG (what you see is what you get) system where as the user interacts with the pages of the Report the interface reflects exactly what the exported document will look like.

The report mainly acts as a form to be filled by the user and that is done in one of three ways. The Auditor can double-click any page and submit, in a pop-up, the pertinent information to that page, as most pages have exact amount and types of data to be filled. Some pages with sections that have free form content, present the user with a text editor similar to a regular word processor like Microsoft Word. Lastly, the Auditor adds or removes Issues and their respective content to the report by searching and selecting in a sidebar that presents itself when needed.

Finally, in the Report Editor interface, there is the possibility of reading or editing comments relevant to a given page. Each page where comments could be relevant has a comment section next to it. The Reviewer reads the report and proceeds to write comments on the necessary pages for the Auditor and later the Auditor uses this comments to correct and refine the document.

4.2. Implementation

To implement our solution in a safe environment and without compromising private client information, we recreated all the existing MAINSEC services and databases as mock-ups. This way we could use the existing functionalities and add to them without putting their service at risk.

4.3. Architecture

Our application follows a two layer structure, front-end and back-end, connected with an API that transfers the necessary data between them. The first layer is the Report Service, a back-end server where we designed the API responsible for making requests to get data from our databases and sending data to be archived in those same databases.

The front-end layer is composed of a client application that uses our GraphQL API to execute the tasks, sent from the user interface, on the back-end server. That same client application is where we developed the user interface using Angular.

4.4. Server - Designing a GrapqhQL API

The backend of our application was implemented using Apollo Server [3], which is a GraphQL server that works with Node.js [1] HTTP server frameworks, in this case we are using Express [7]. Coupled with our Apollo Server, we used two ORM technologies, Sequelize [15] and Mongoose [12], to help us map the objects from the PostgreSQL and MongoDB databases.

With all the tools mentioned above, we developed our service using three distinct components to manage each object involved in our application. These components are the schema, the models and the resolvers, which we will explain below.

We begin with our schema files. These declare all object types and possible functions regarding these objects, using GraphQL schema. This means that for a given object, like a Project, we defined what are its features (name, status, date, password, etc.) and what actions the application can execute on these objects (create, read, update, delete). As shown in section 4.5.

Secondly, we create the models. In these files we define the type of each feature in an object, i.e., the name of an object would be saved as a string, and a list of Reports in a Project would be mapped to an array. The way we produce the model files depends on what ORM technology is being used to match each database. Models for objects stored in a PostgreSQL database are defined using Sequelize, and Mongoose is used for MongoDB database systems.

Lastly, resolver files contain the implementation of the functions that we can use to access and manipulate the data stored in our objects. For example, to change a review deadline of a Report, we have to input the Report id and the desired new deadline, then the service will return the updated Report object to be presented in the interface. As developing models differ depending on the target database, we have to use both Sequelize and Mongoose to implement how the CRUD functions are executed in their target database.

The complete service definition enables the client-side application to interact with the objects and their functions using only the generic GraphQL schema to make calls to the service. The service, then, internally executes the functions using their defined resolvers, without the user needing to know how each database works.

4.5. Main Object Types

Following the requirements, in Section 3.2, we define different objects for Projects, Reports and Issues, respecting all their features. We also created new objects and features which we will now describe.

In our implementation, we split Report into two objects. The first represents the report in the Report Repository, only containing meta information necessary for the Users to manage and select it for viewing or editing. The Report Document object defines how the document and its contents are stored in the MongoDB database system.

We defined each User as a basic User which only starts with its login information('name','email' and 'password'). As it is added to a Project as Auditor, Reviewer or Project Manager, the User object is extended with a 'role' field, affecting the way the application behaves.

A Client object was created by copying the User object and removing 'password' field. As there is no intention for this entity to interact with our program.

We added an 'id' to every object that was automatically generated by the respective database upon creation of an object. PostgreSQL was responsible for Projects, Reports, Users and Clients, while MongoDB holds the Report Document and Issue objects.

We also added other 'id' fields inside related objects as a way to easily interchange data between objects. For example, a Report Document gets information about its attached team members by using the 'projectId' of its parent Project.

The Report Document object is composed of multiple complex objects that represent its sections filled with static and dynamic data. It also has an array of Issue objects to store the Auditor added Issues. Finally, there is a 'reviewerComments' object that has comments added by the Reviewer separate by sections.

The Issue object has fields already full of static content from the Knowledge Base, and is added to the Report with 'technicalDetails' and 'currentStatus' empty fields to be written by the Auditor. There is also and array of 'IssueFigures' that is used to store URLs and captions to appended images. finally, each issue keeps its own Reviewer generated comment in a 'reviewerComments' field.

4.6. Client - Using the GraphQL API

For the client application to get, manipulate and save data in our databases, we had to write functions to interact with the GraphQL API implemented in our server application. As it would be a taxing task to manually create client-side functions for every action possible in the API, we used the tool GraphQL Code Generator [8] to automatically generate the desired operations.

Automatic saving of a Report is possible, because the input of new information to the current document is done though the GraphQL API and each call only fills a portion of the document. This means, that filling a single page with content equals an automatic save, instead of saving an entire document at a time.

4.7. Client - Web Interface

In this section, we will demonstrate, through the help of figures, how the principal interface components and features were implemented and how the User interacts with them to accomplish the tasks proposed at the beginning of this thesis.

As a whole, the interface was developed using Angular. We chose Angular Material [2] as the design signature of our application.

Login

The first interaction the User has with the application is the login screen, where the system takes the User information to check if he can use the program, and to select the correct data to present to the User. The login information is submitted through a simple form.

Report Repository

As the User initiates a session he is presented with a list of Projects he is assigned to. These Projects can be filtered with the search bar on the top, and can be sorted by clicking one of its features. Searching and sorting capabilities are present in all other tabs as well.

From the Projects tab, any User can create a new Project. However, only a Project Manager can edit the meta information of a Project or delete it. This is demonstrated by the available buttons on the right. Alex only is the Project Manager in Project A.

When the "Create" Button is pressed, a dialog will appear. Then, the User fills up the information accordingly. This is helped by the fact that, apart form the Project name, all fields only show information that respect the rules imposed by MAINSEC. Only a predefined status can be select, and the possible options in the Client, Auditor, Reviewer and Project Manager fields are entities that already exist in the connected databases. The same dialog is used to edit the Project information after it is created. To open a Report for viewing or editing, a User double-clicks the name of a given Report.

In the Reports tab, once again, a User only gets to see the Reports belonging to a Project he belongs to. Any User can create, edit or remove a Report inside a Project he is assigned to. Reports can also be opened in the Reports tab. Creating and editing a Report uses the same style of form used to create a new Project.

The Clients and Users tabs show a list of all the Clients and Users to every User, as these function mainly as a contact book. Only the Project Managers can edit or delete elements of this list, but all Users can add Clients and Users as needed.

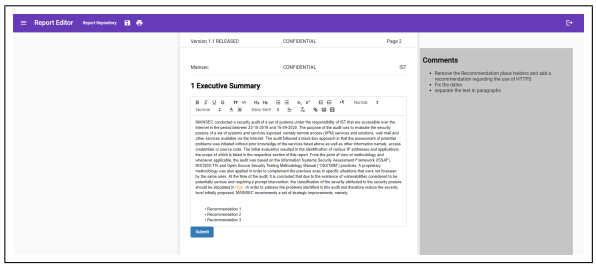


Figure 1: Editing content in a section using a rich text editor.

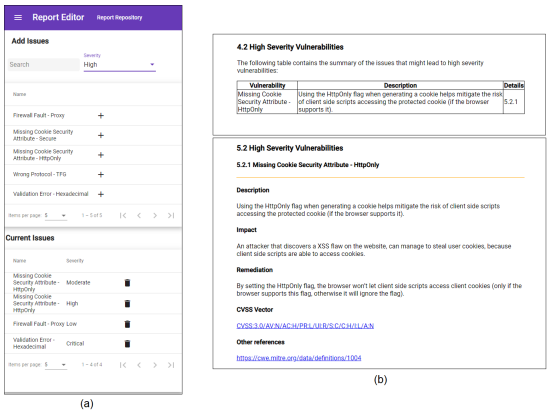
Report Editor

After the User selects a Report to open in the Report Repository, the interface changes to a WYSIWYG paradigm. A User can scroll to see how the Report will look like when exported as PDF, and double-click in a page to be edited.

As a target page is double-clicked, the correct tool to edit the present contents will present itself. Pages are edited either by filling form or by writing in a rich text editor, shown in Figure 1. In our application, we chose Quill [14] as our text editor module, as it is easy to use and implement. Additionally, Quill can save the edited text, internally, as HTML. The HTML content produced by Quill has the advantage of being lightweight and fast to store in String format inside our database, and later on be read and reproduced to the interface just as

easily.

On the right side of Figure 1, there is the comment section. This section shows a different behavior depending on the team member in session. The Auditor and Project Manager can only read the comments left by the Reviewer, while the Reviewer interacts with a text editor, similar to the one seen in Figure 1, to submit his comments on that page. This distinction is possible, because when the User opens a Report the application is aware of the User role in the respective Project.



(a) (b)

Figure 2: Add an Issue to the Report. (a) - Issues sidebar. (b) - Issue automatically placed in the correct sections.

Add Issue to the Report

To manage the Issues in the Report, the Auditor must open the hidden sidebar by clicking the menu button on the top left corner. In the first table in the sidebar, shown in Figure 2 - (a), Issues from the Knowledge Base (section 3.6) can be searched and sorted. In the table below, the Auditor can remove Issues already in the Report by clicking in the trash can button of the target Issue.

To add an Issue to the document, one must first select a Severity from the selection element next to the search box. Then, after clicking the plus sign in the desired Issue, all the static data of that Issue will be automatically imported from the database to the to the Report.

Figure 2 - (b) shows the results of adding and Issue to the Report. First, a summary of the Issue is inserted in a table of the select severity section, including an indication to the full description in the next chapter. Second, a detailed definition of the Issue in the Assessment Details section.

Editing an Issue

After an Issue is inserted into the document, the Auditor has to complete it with the Technical Details and Current Status paragraphs. To do this, the writer double-clicks the recently created page with just the two titles of the paragraphs. In the pop-up dialog, paragraphs can be written in the left text editor and images can be added through the form on the right.

5. Evaluation

To test the usability of the system proposed in this document, we compared it to the workflow in place at MAINSEC. In this test, we have the volunteers execute a set of tasks in a Google Docs document and in our solution.

This chapter describes the study that took place to evaluate both solutions.

6. Participants

The tests where done by eight volunteers, six men and two women. All of our participants work in IT companies as developers and/or managers and everyone has experience in creating report documents.

7. Procedure and Apparatus

The sessions had an approximate duration of 30 minutes and were done in a calm and quiet room with a desktop computer. Each session was initiated with an explanation of how the tests would take place and what they would be doing.

First, we explained the window layout in the two monitors. The first monitor had the interface to be tested and the second one had the instructions document for the tasks, as well as an auxiliary document containing the information for the Issue to be added to the Report.

As everybody had previous experience with Google Docs, and the instructions given provided all the necessary knowledge to use SAAR2020, no time was invested in learning the interfaces.

We informed them that there were two questionnaires at the end of the tasks. Finally, we explained that their results would be kept anonymous. Any further questions were welcome to be posed during the test.

Then the testers started performing the asked tasks by following the instructions given and we started taking note of the time taken with our stopwatch.

Each user performed the same seven tasks, listed below, on both applications. Half of the users started with Google Docs and the other half started with SAAR2020, so that the results would not be affected by the user learning the tasks. The target document already had some comment and only had to be modified.

T1: Edit the document cover by changing two words and a date, in the correct format;

T2: Edit the headers by changing two words;

T3: Add three names to the Auditors and Reviewers table;

T4: Add a new row to the Document Management table and fill it with 3 words and a date, in the correct format;

T5: Add an Issue to the document using the provided Issue information and place it in two sections, each one with its own formatting;

T6: Edit the dynamic data of the added Issue by writing two paragraphs and adding a captioned image;

T7: Export the document as a PDF file;

At the end of the task execution on both solutions, we asked the participants to rate each task on each solution using a Likert scale (1=not at all, 10=very) regarding how fast and how easy it was to do the task and how useful the interface was in helping the user perform the task.

8. Results

In this section, we present the results of the comparison between the two solutions. The results obtained from the two solutions are described by the metric and compared with each other.

8.1. Time taken to execute tasks

The time (seconds) a tester took to execute the tasks was measured from the first click on the interface to the end of the last task.

We saw some difference in the duration of test completion amongst users. This happens as different participants have different levels of experience with the technologies used during the tests.

Even though the testers took more or less time performing tasks, we could verify a consistency in the ratio between time taken on each solution. With SAAR2020 times being, on average, 2.59 times faster than Google Drive. The User registering the smallest difference at 2.46 speed improvement and the biggest at 2.73 times.

8.2. Satisfaction

In Figure 3, we show the average score attributed in the three questions posed about each task. These scores were obtained by having the testers answer a questionnaire for each solution. These questionnaires were answered using a Likert scale. Every Likert scale has ten discrete, from 1 to 10 points. Score 1 represents, for example, "the task was not at all easy to do", and score 10 represents the reverse, "the task was very easy to do".

Overall, we noticed an improved or constant score from Google Drive to SAAR2020. This happens, because at every step our application tries to reduce the input required from a user. This is especially noticeable in T2 and T3, where SAAR2020 got almost perfect scores, because both these steps are done automatically by the application itself in the background. T5 and T6 received in Google Docs received the worst scores as these are the tasks that required the most user interaction.

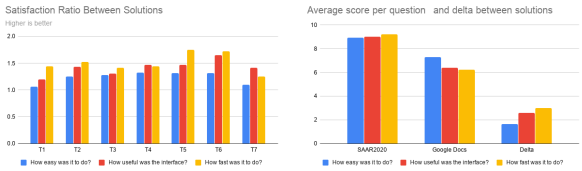
We also calculated of the satisfaction ratio per task between the two solutions, in Figure 4 (a).

(a) Google Docs (b) SAAR2020

Figure 3: Analysis of the time taken the execute the tasks.

And, then, averaged the improvement ratios of all the tasks per solution and measured the difference (Figure 4 (b).

The difficulty was the least affected, receiving 1.64 more score points in SAAR2020. The usefulness improved in a greater manner, getting 2.59 more points than Google Docs. Finally, the best results came from the enhancement in time taken to fulfill the tasks. Users gave 3 more points, on average, to SAAR2020.



(a) Satisfaction per task (b) Average Satisfaction

Figure 4: (a) - Satisfaction Ratio Between Solutions. (b) - Average score per question and the difference between those averages.

9. Conclusion and Future work

Presented with the challenges faced when creating reports manually, we proposed SAAR2020, an automated report creation system that enables the users to only invest their time writing new data instead of copying existing information form one place to another.

We compared this solution with a traditional word processor, Google Docs. Results show that SAAR2020 is over 2 times faster than Google Docs and users find this solution more useful and easy to use. Improvements to the application can be made by further tuning which tasks can be automated and adding new features as found necessary.

References

[1] About — Node.js.

[2] Angular Material UI component library.

[3] apollo-server-express - npm.

[4] Cloud Storage for Work and Home - Google Drive.

[5] Comparison with Other Frameworks — Vue.js.

[6] Definition of REPORT. *www.merriam-webster.com*.

[7] Express - Node.js web application framework.

[8] GraphQL Code Generator — GraphQL Code Generator.

[9] Guide: What is Vue.js? *Vue.js*.

[10] Introduction to LaTeX.

[11] Introduction — Vue.js. *vuejs.org*.

[12] Mongoose ODM v5.10.5.

[13] Overleaf - About us. *Overleaf*.

[14] Quill - Your powerful rich text editor.

[15] Sequelize — Sequelize ORM.

[16] Write papers like a modern scientist (use Overleaf or Google Docs + Paperpile). *Simply Statistics blog*.

[17] Facebook Inc. Introduction to GraphQL — GraphQL, 2019.

[18] Google. Angular - What is Angular?, 2019.

[19] Ilya Bodrov-Krukowski. Angular Introduction: What It Is, and Why You Should Use It — SitePoint, 2018.

[20] P. Madan. *Language proficiency in English*. Agarwal publication, 28/115, jyoti block, sanjay place, Agra-2, 2016.

[21] I. MongoDB. What Is MongoDB? — MongoDB, 2018.

[22] SAP SE. Analytics for the small to midsize business with SAP Crystal solutions SAP Solution Brief SAP Solutions for Small, Midsize Businesses. Technical report, 2018.

[23] SAP SE. SAP Crystal Reports reporting and analytics solution, 2018.

[24] The PostgreSQL Global Development Group. PostgreSQL: About.

[25] TIBCO Software Inc. Jaspersoft® Studio — Jaspersoft Community, 2018.