M1 MIASHS IC

Project report

TER Project

2021 - 2022

Descotils Juliette
Gattaciecca Bastien

1. Introduction

1.1. Purpose and context

The computer development project described in this document is part of the TER project (“Travaux Encadrés de Recherche” : supervised research project) of the Master MIASHS Informatique et Cognition.
Francis Jambon and Philippe Mulhem are researchers in computer science at the Laboratoire d'Informatique de Grenoble (LIG) and have proposed, in collaboration with students of the Master, a system aiming at improving and fluidifying techniques used in WEB research. Indeed, WEB research represents today a major stake for most companies, and a system aiming at making it more effective is very much sought after.

1.2. Motivation and stakes

Francis Jambon and Philippe Mulhem are trying to improve some techniques used in WEB search.To do so, they have developed a sophisticated search engine based on the Terrier engine and a results database as well. This database is able to return relevant results to the user for the query he has sent. At the moment, the relevance of the results is measured using an eye-tracker. Francis Jambon and Philippe Mulhem have conducted a study in which they show that there is an advantage to using the user's gaze as an implicit measure of the relevance of the words proposed in the results [Sungeelee, Jambon, Mulhem, 2020].

1.3. Presentation

In order to improve their search engine, Francis Jambon and Philippe Mulhem aim to have the relevance of the results evaluated by experts in the query domain. The request is to develop an ergonomic graphical interface to allow them to quickly judge the relevance of the results of a query. An existing application, named "Relevation!" [B. Koopman and G. Zuccon, 2014], dedicated to the same problem, has already been implemented and used in the past.
However, the Relevation! interface has limitations and is outdated. The technologies used are not up to date; and it is not directly usable in our case, as the notions of **experts** and **snippet** are not present.
So, it is necessary to create our own software, more robust, scalable, and up to date compared to Relevation! It will have to allow experts to annotate snippets and documents returned by the developed search engine according to a topic. By taking into account the expert review on the returned results, our two supervisors will have the necessary data to measure the relevance of their search engine.
The project lasts for the whole year of the Master. The first part consists in doing the analysis and choosing the technology we will be using. It takes place from September to May on a part-time basis. The second part is about programming and making the internal documentation as a full-time job at the IMAG building.

2. Development

2.1. Project management

2.1.1. Logbook

Throughout the year, we took notes during the meetings in a logbook. This allowed us to efficiently summarize every review after each meeting. We also put our interrogations and the questions to ask for the next meeting. This document has been of great help to us in writing the reports of the meetings as well as in writing this report.

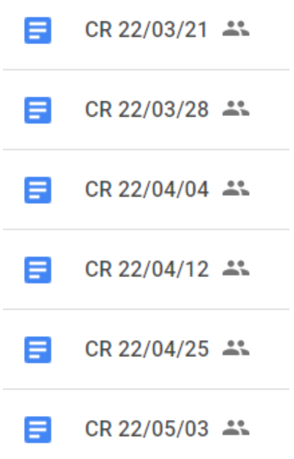
Journal de bord

2.1.2. Reports of meetings

Reviews are taken after each meeting. Their structure is always the same:

- Date and participants: indicates the day/hour start time of the meeting and who is present during the meeting.
- What has been said: indicates in the form of several dashes the content of the meeting (what was discussed about).
- What remains to be done: indicates in the form of several dashes the remaining tasks. The goal is to complete all of them before the next meeting.

All of them are written and put on the git wiki page.


CR 22/03/21
CR 22/03/28
CR 22/04/04
CR 22/04/12
CR 22/04/25
CR 22/05/03

2.1.3. Weekly meetings with our supervisors

We had at least one meeting per week (excluding holidays) in the IMAG building. Each meeting consisted of a summary of our progress since the last meeting. This was followed by a review of our work and some changes to be made.

21
● 10:00 RDV TER

2.1.4. The development environment

We used Google Drive for file sharing and collaborative writing between us. We also had shared files with our supervisors. All our code was written on Visual Studio Code. The client code was done separately on a browser to do the tests. The PHP code has been tested on OneCompiler by small parts before being added in the global code. This was especially useful for unit tests.

2.1.5. Git management

We used git as a versioning application from the Git extension of VScode. We did not use any other branch than Master, this allowed us to manage all the merge conflicts as soon as they came up. We both pushed our code at a frequency of once or twice a day ; usually at the end of the day at the IMAG office so that we could keep working at home in the evening. The main use of Git for us was the history function to be able to retrieve some code that was made and then replaced. Another advantage was to be able to add files in the .gitignore so that we do not have to change local parameters (like the path to the database).

2.2. Analysis & Design

2.2.1. Technical and functional specifications

The specifications contain the functional and technical requirements as well as the constraints and the available resources for this project. This was a relevant document for the first part of the project which was only to define the criteria to be met and the list of all necessary and optional features of the application. As the project is in a development phase and the specifications must be more accurate, the Pages and Files Specifications was the document to be referred to. First, this document contains all the interface details, and allows the design of the first mockups of the interface. Every interface element is listed and has a graphical description as well as the triggered events. In addition to that, the document also specifies the structure of all files used by the application which is an important thing to know as coding readers that will parse those files.

2.2.2. Technologies used

The biggest part of the project is the programming of the software features, which was done in php. This language allows to create dynamic web applications articulated with a database. The quantity of methods and functionalities to be implemented being very important, it is widespread to use a framework allowing to generate automatically certain functionalities and a project structure. This is why we chose to use Laravel which is an open-source web framework written in PHP based on the model-view-controller principle. This framework is widely used today because it is well documented and maintained. It allows to set up a secure authentication system and to reduce the development time and the addition of future features.

The development of the interface was done using HTML to represent the content of the different web pages of the application. Thus, the different views of the application (coded in .blade.php files as implied by the use of Laravel) are written using HTML and are articulated with files written in CSS. The CSS language allows the creation of style sheets that can be attached to the different views in order to customize the graphical aspects of the interface. These two languages have been chosen because they are very widespread in the world of web development (if not the only ones existing) and because they are well documented. Moreover, they are well mastered by the developers of the first version of this application which allowed to reduce the development time.

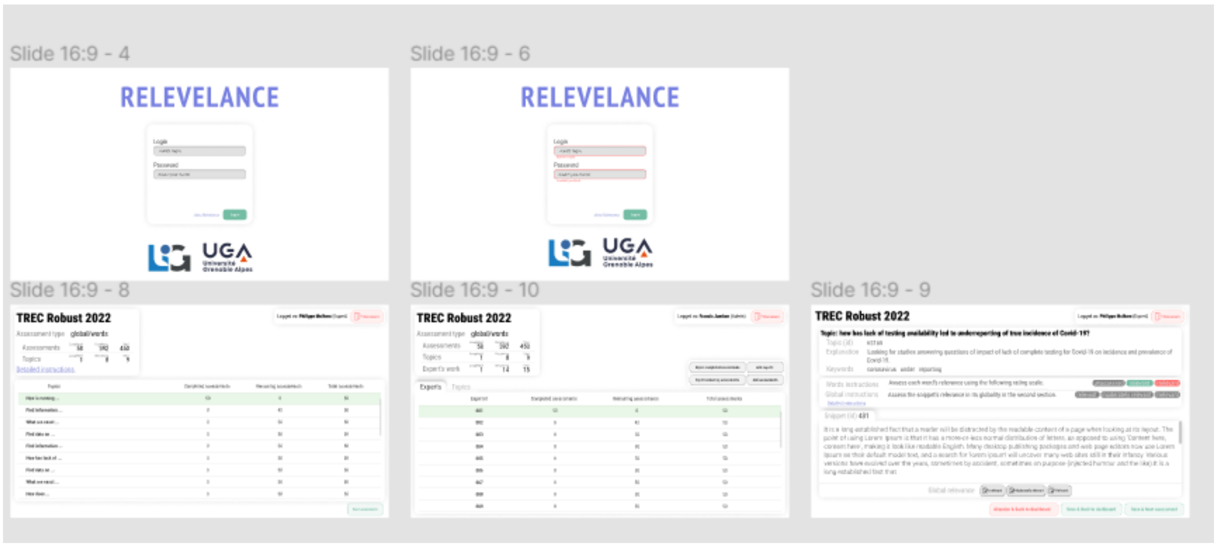
In order to create an interactive web application and in particular to update the information displayed in the user dashboards, it is common to use JavaScript. JavaScript is a language allowing to manipulate the objects of a web page in a dynamic way, it can be used for back-end and front-end development but in our case it is only used for the front-end. It is used by respecting the model-view-controller principle. It is also common to use a JavaScript language framework when developing applications with a high degree of complexity at the interface level. It was therefore initially planned to use Vue.JS which is one of the three main frameworks for JavaScript. This one had been chosen for its reputation of being a complete, well-maintained and easy-to-access framework for beginners. However, we quickly realized that the development of the user interface was relatively simple and that using JavaScript would suffice. Indeed, concentrating on the CSS of the different views and taking care of the visual aspect took more time than the implementation of the interactivity of the tables present on the dashboards of the administrator and the expert dashboards. In order to interact with HTML code efficiently, the jQuery library provides methods to easily manipulate DOM objects and handle events.

In order to store data related to users, annotations, endorsements and the campaign, it is necessary to structure the information. For this purpose, it is possible to use files or a database. At first, we chose to limit ourselves to the use of files, but a database quickly proved to be more efficient in order to structure the data in a table format. The database allows faster access to the stored information. Moreover, it is more usual to use a database to centralize and organize the data storage. Since the amount of tables and variables to be stored is not very large, a simple database is sufficient. SQLite is a library that implements a simple, fast and sufficiently reliable SQL database engine for the application's use. It allows to store input files written in XML or JSON and to access the information through classical SQL queries.

For the choice of the incoming and outgoing file formats of the application, it was simple to refer to the formats generally used in the field. Thus, the files intended to be exported by the administrators are generally space delimited text files that can easily be used or parsed to another format. The incoming file are the files containing the lists of administrators, experts, documents, snippets, topic and the campaign file. The format generally used is XML or space delimited text. The topics are usually described in TREC format files which are easily changeable to XML format.

2.2.3. Mockups and interface

We presented several versions of mockups to our supervisors. The main elements of the interface were detailed in the Page and Files Specifications. A first version of the interface mockup could be proposed. But as all the details of the interface could not be detailed in the document, it took us 5 iterations for our supervisors to validate the mockup.
The first mockup version was made on Google Draw. We switched to Figma quite quickly as it is a dedicated application for interface design and development. We presented a dark theme in addition to the main white theme but due to lack of time we could not implement it. Indeed, since the goal of the application was to provide an ergonomic interface, we thought that adding a theme preference for the user was a good idea. However, such a feature would have required changing other colors that are useful to the application such as the background of the annotated words which would have a bad contrast. We have made four views on Figma. They are available in the annex. The dark theme versions are also present, but at a less advanced level than the white theme versions (which are the final versions) since, as mentioned before, they have not been implemented.



2.2.4. Software structure

The choice of the software architecture is very important to define how the modules are organized among themselves and at which level a functionality should be defined. A software architecture well built and adapted to the need allows to reduce the development cost and to increase the quality of the software. Indeed, it is easier to maintain a software with an organized and logical architecture. It also allows to reuse software components and thus to save money during development. A clear and organized software architecture also allows to improve the quality of the application on the main criteria, defined during the phase of analysis and writing of the specifications such as maintainability, reusability, compatibility and reliability.
The MVC architecture (model-view-controller) is commonly used to develop web applications with graphical interfaces. It allows to separate in three (or even 4) categories the modules according to their responsibilities: the model gathers the modules concerning the logical data, the view contains the modules of management of the graphical interface, the controller organizes the interfaces used by the model and is controlled by the controller
This software architecture being the one we have studied the most and with which we are the most comfortable, we chose to adopt this one.
The model part gathers all the methods allowing to access the information stored in the functional core (getters) like the definitions of the expert, admin, snippet, assignment objects etc. It also contains a reader class and a writer class for each type of object which contains methods to read incoming files and to write export files.
The view part itself is implemented by creating the following three modules: one for the connection view, one for the dashboard view, and one for the assessment view. The most important modules requiring the most methods (the dashboard view and thee assessment view) have been implemented using the MVC principle as well. Thus, for each of these two views we separate the methods managing the content to be displayed (in the model), the methods managing the way of displaying the information (in the view) and the methods allowing to update this content automatically (in the controller).
The controller part contains the methods to perform actions such as logging out, saving an assessment, returning to the dashboard or displaying the next assessment. It allows you to modify the information in the database via the methods of the model and to modify the display by redirecting to other views.

2.2.5. Data model

Before coding the server backend, we first had to define all the methods that would be needed in a document. For this, we used the final version of the mockups that shows directly the data that must be retrieved from the DB. We also proceeded by writing the methods that directly answer the requested functionalities. But since they are at a too high level of granularity, we deduced more basic methods that would be dedicated to these higher level methods. The interest of splitting the code is to gain in readability and to facilitate unit tests.
Then we grouped the methods by granularity. For example, the method that fills a table is at a higher granularity level than the one that allows you to read a file since it includes it.
Finally, all the methods have been grouped into classes according to our data model. We have made two of them.

The first one was an actor-oriented model where the classes represented the different actors of our model with instance methods. For example, there were Expert and Admin classes with instance methods that allowed them to interact with other objects. The problem with this method is that you have to change the model and the base at the same time and therefore can create consistency problems if one is modified and not the other. A priori it can be done with a little rigor, but we see two technical problems: the first is that an actor-oriented model makes a lot of method calls for interaction between objects, which leads to making a lot of requests to the database (at least, more than necessary), which is a performance problem. The second problem is if you make a modification in the model and in the database and one of the two instructions makes an error (and not the other). For example, the model can accept to have two experts with the same identifier where the primary key of the experts table would not allow it in the database. This can lead to consistency problems between the database and the model, even though the same modification instruction was made in both.

The second is the task model. This is the data model that we have chosen. The task model presents an efficient interface of the database in the model so that each class represents a table in the database. Thus, the class methods are simply retrieving information (select) or making a change (add/delete/update) in the database. Consistency is more secure: the model always gives feedback on what is happening in the DB. The performance problem is also solved as only the necessary calls to the DB are made.

The data model of the DB has been schematized as presented in the appendix. We have chosen to make it a schema rather than an exhaustive specification document since database management systems have many well defined conventions. So a simple star on a column showing that it is a primary key is enough to indicate something very specific.

2.2.6. Annotation methods

One of the main functionalities of the application is the annotation of the relevance of words in documents and snippets. Part of the design phase of the application was therefore to think about how to implement this functionality. We started to think about this issue from the second meeting with the supervisors and discussed it among ourselves. At first, the solution adopted was to click on a relevance level after having clicked on the word to annotate. This solution is used by other applications but is not very efficient and requires a lot of clicks and mouse movements. This can be costly and the work of the experts is often composed of many documents and snippets to annotate on many topics. It is therefore important to try to minimize these two parameters in order to minimize the annotation time and thus the work time of future users. We quickly considered clicking on the word directly to minimize the mouse path and to allow visualizing the change of relevance thanks to a change of color of the word according to the assigned level. The supervisors then made a similar proposal which allowed us to choose this option.
Later, when the application was much more advanced, we discussed the possibility of changing the relevance level using the keyboard keys to reduce the number of clicks. Several proposals were made here as well: one key to select a word (tab key) and one to increment the relevance level (enter key), two keys to select the word (keyboard arrows) and as many keys as relevance levels (starting at the first letter key on the keyboard), two keys to select the word to annotate (keyboard arrows) and one key to increment the relevance level (tab key).
The first solution was the first one implemented. Unfortunately the tab key is also used to select the clickable fields of the browser, so it is more complicated to use this key, so we chose to replace it by the 'a' key, located right next to it on the AZERTY keyboards. After discussion, the possibility to go back to the previous word seemed important, so we chose to add the possibility to go back with the left arrow key.
To annotate a word on the application, you can either click directly on the word as many times as you want to incrementally increase the relevance level, or, from the already selected word (the first one if no click has been made) navigate to the next or previous word thanks to the keyboard arrows and then press the 'a' key to increment the relevance level. When the maximum level of relevance according to the definition of the campaign is reached, it changes back to 'not annotated'.

2.3. Implementation & programming

2.3.1. Database

The database schema in the annex shows the different tables of the DB with their names. A table is represented by a table in the diagram where each row corresponds to a column in the table. It is defined by a name, a data type (date, integer, string), a

constraint (dependency or a constraint on the type as different from null or null accepted) and a star that indicates if this column is a primary key of the table or not. If there are several stars in a table, then it is a composite key.
The arrows between the tables represent the dependencies between them when an attribute of one table references the attribute of another table.
Concerning the structure of the DB, we have chosen to separate the experts and the admins in two different tables. The first reason is the fact that the expert table can be modified during the campaign whereas the admin table cannot (it is defined at the start of the campaign). The second reason is the fact that users can connect either as an expert or as an admin, so if we had put experts and admins in the same table, we would have needed a third column indicating the role associated with the login/password.
The documents table has two columns external_url and internal_uri which specify a path to access the document. For a record in this table, if the url attribute is set, then the uri attribute is not, and vice versa. So one of the two attributes can be null but never both at the same time.
To remain consistent in our choice to use a database and not files, we have added a campaigns table which, presumably, will contain only one record.
Concerning the snippets table, it has a composite topic/document key only (no id as primary key) because we know that a snippet is a part of a document chosen according to the topic. So we are sure that for the same topic, the search engine will always select the same portion of text in the document to create the snippet; hence the uselessness of having a primary key.
The assignments table is a pivot table that allows you to obtain a primary key from a composite key of three elements. An assignment designates an expert who annotates a document under a topic. In the *_assessments tables, we need to know which assignment this annotation refers to. Rather than repeat three columns in the *_assessments tables to indicate which assignment it is, we prefer to retrieve the unique ID of the assignment. This reduces the information load contained in the tables.

2.3.2. Session management and user connection

Our application allows to manage the identification of users. There are two types of users: experts and administrators.
Both users have access to their dashboard upon login. However, administrators have additional functionality to manage the campaign while experts have a functionality to annotate documents/snippets. Note that there is no registration. It is only possible to log in. Indeed, the identifiers are given to the administrators and the experts at the beginning of the campaign. They are therefore added to the database beforehand: there is no need to register. This also avoids having to provide an email (to retrieve the password or other) which allows to store less personal information in the database.
We have chosen to implement two different logins for experts and admins. This allows to know, before he enters his login, if the user wants to connect as an expert or as an admin. Indeed, it is possible that a person is both an expert and an admin at the same time. So to know if the provided login should be searched in the expert or admin table, we have created two different login pages.
The session system is managed by adding a token in the user's browser. Thus, the browser knows if a user is already logged in or not. This has an effect on the following cases:
- logging in

If the user is already logged in and opens a second tab, he can access his dashboard with the url /dashboard immediately. If he decides to go through the welcome page (and not directly through /dashboard), then he will be directly redirected to his dashboard without having to re-enter his login information. If the user is not logged in, he will be redirected to the /welcome page to enter his credentials.
- Log out

If the user is already logged in on several tabs, logging out on one tab will log out on all the others because the token dropped in the browser is removed. However, it will be necessary to refresh the page so that the disconnection redirection is done.
When the redirection to the welcome page is done, it is not possible to go back with the browser backspace.

2.3.3. Client/server communication

For the sake of flexibility, it was chosen that the smallest granularity was the assessment. This means that an assessment is done locally on the client side, and there is no communication between the server and the client during the annotation of an assessment. This implies that an assessment cannot be saved during the work to come back to it, and that a request is sent to the server only to retrieve the assessment and send it when the expert has finished his work. This makes it possible to put all the information in a single variable and send it in the stream rather than having to make several requests. As explained above, the annotation of a document/snippet is coded in html/js. So when displaying the assessment view, we had to retrieve variables from the server to become javascript variables and set the parameters of the model. To do this, we transformed these php variables into JSON content so that it could be parsed by Javascript and then recovered its content as in a JS object. Similarly, in the opposite direction, when saving an assessment, we put all our data in an object that we transform into JSON content and then send it to the server via a POST method. We made this choice to rely on the usefulness of the php methods json_encode and json_decode which are methods that encode/decode strings in JSON format efficiently.

2.3.4. Logs

For all the methods that only make an effective modification of attribute(s), we have chosen to make them return a boolean that indicates if everything went as planned. This allows us to better manage the exceptions that are raised. For example :
getNextAssignment() is a method that returns the next assignment of an expert via a query that retrieves information from the table. Rather than throwing an error when the query fails and there is no next assignment, the method simply returns false.
We chose to use a log file at the root of the server in which a line is written each time an exception is caught and the method returns false. Each line contains a timestamp and a message detailing the error in question. Indeed, the advantage of writing a line in the log file in the context of the error allows to have a lot of information and thus to facilitate the tests.
Furthermore, these error messages were very useful for testing methods that are dependent on others. Indeed, by simply looking at the message stack in the logs, we could determine where in the code the error came from.
In any case, this was a suggestion from our supervisors during our meetings that seemed important to allow us to have a display of errors as managers of the server.

2.3.5. Documentation & Tests

The scalability and robustness of the application are key elements of this project and this demand was specified at the beginning of the specifications writing. To ensure the scalability of the application it is essential to produce a complete and accurate documentation of the code. Thus, throughout the development phase, explanations on the objectives and functioning of each method were written. This also helped us during the programming phase to efficiently find the objective of certain modules that we had programmed long before or to understand a piece of code written by the other developer. The implementation of tests for each method and each module of the code is necessary to ensure the robustness of the application. Each method was tested several times to ensure that each of its objectives was met. For example, a method that was supposed to return a view or an error was tested in each condition immediately during its development. In addition, tests were performed when several methods were operating together to ensure that they were correctly articulated. Finally, when the code to test an entire feature was completed, a series of tests covering as many outcomes as possible was performed. Presenting a completed module to the other developer and then to the project managers also allowed for some errors to be encountered and some bugs to be fixed.
A part of the tests performed throughout the project is written and explained in order to be able to reproduce them in the "test plan" document.

2.3.6. Readers

Readers are classes in which functions related to the reading of files imported into the application are defined. A reader class is defined by object corresponding to a table in the database. Thus we find the AdminReader, AssignmentReader, CampaignReader, DocumentReader, ExpertReader, FileReader, GlobalTagReader, SnippetReader, TopicReader and WordTagReader classes. All these classes (except FileReader) redefine the parseFile function which has the following objective.
The AssignmentReader and ExpertReader classes also redefine the parseFileFromContent method which takes as parameter a string and not the path to a file. These two classes have this particularity because experts and assignments are the only objects that can be added to the database by the administrators. These two methods are therefore used to retrieve data from files written by the administrator and not loaded at the beginning of the campaign. Just like the parseFile methods, they return an array containing the information of the file distributed in the different cells of the array.

2.3.7. Modals

Before recording an assessment, the initiative was to add a modal window to check that the assessor actually wanted to validate the assessment as there is no possibility to go back once the action has been completed. This proposal was not retained as it implies an additional mouse move and click for the expert.

3. Review

3.1. Critics and remarks

Another thing we didn't think about was writing tests. Although we tested all of our methods and features as we developed them, we didn't think about writing the test flow. So writing the test plan was a lot of work to do all at once. The quality and completeness of this document would probably have been better if we had written it at the same time as we were doing the tests.
Furthermore, we had planned to start writing the documents after finishing the source code. This seemed feasible given the lead time we had taken for development. Indeed, only a few details remained to be developed at the date we had set for the end of the development but we were surprised by the time it took us to develop these details. Finally, we started writing the explanatory documents a few days late and this forced us to finish the writing in a short time.
Finally, we sometimes spent time on tasks that did not serve us in the end. For example, we thought we needed to formulate SQL queries to access each of the data in the database and therefore spent at least two days of the development phase formulating these queries that we did not use in the end.
Overall we made mistakes when estimating the time needed for the different tasks as well as when estimating the need for the modules to be developed. This may be due to our lack of experience in IT project management, but a retrospective phase allowed us to identify these errors and to understand how to avoid them in the future, which seems beneficial to us.

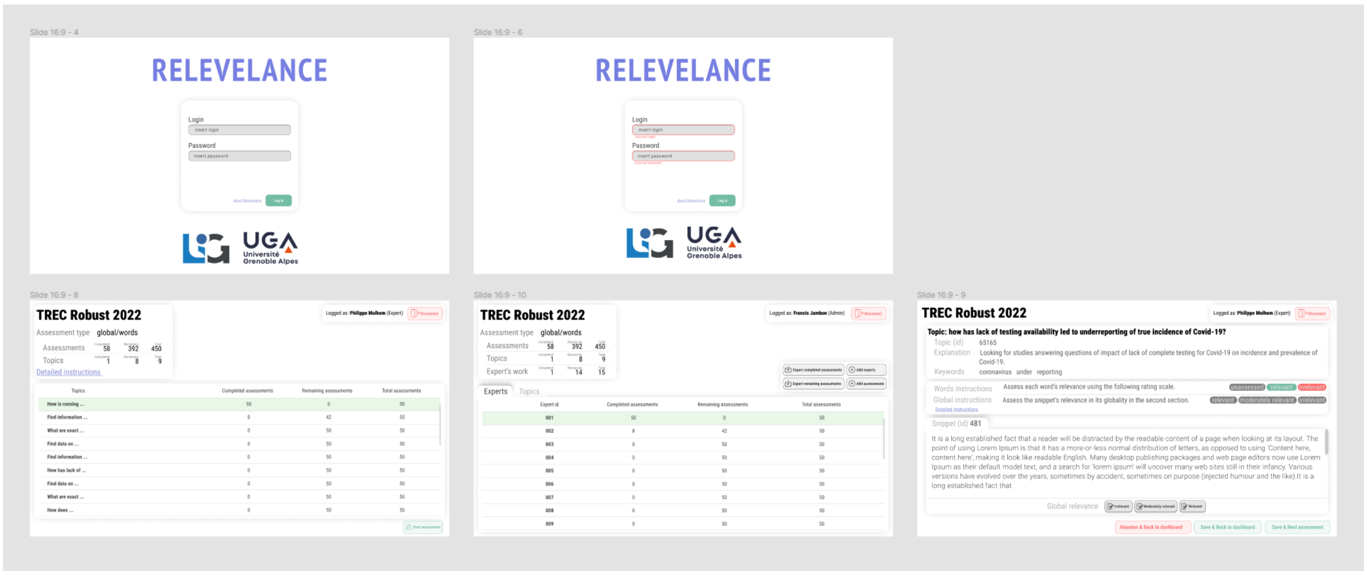
3.2. Possible openings

The rendered product seems to be satisfactory and meet the expectations specified in the specifications, however, we know that other features could have been implemented to make the application more complete. Several options were discussed during the multiple meetings with the project supervisors and could be used to develop the application in the future. First of all, we quickly made the choice to save the modifications made to an assessment only after the user clicks on save. Allowing the user not to save the assessment and to be able to come back to it later would probably be an interesting feature. Continuous saving would facilitate the work of the experts.
We have also chosen not to allow the user to modify an assessment that has already been saved. This made the development work very difficult and the supervisors did not seem to find this option necessary. However, it is certain that experts can change their mind about the relevance of a document or a snippet and it would seem logical that they can access the assessment page again to modify their annotations.
The supervisors mentioned the interest of being able to add comments to each assessment. Indeed, experts may have details to add when they make choices they are not sure of. Adding a comment area on the assessment page seems to be a feature that could help experts and that it would be interesting to implement in the future.
When an expert clicks on save and the assessment is incomplete, the application will generate an error in the logs. Only, the assessment page is refresh and all changes are lost. In the current configuration of our code, it is impossible to generate an alert for incomplete annotations because the verification is done on the server. A much better configuration would have been to do the check locally in the javascript code, and then send the information to the server only if it is correct. If it is incorrect, we would simply pop an alert on the browser that the annotation is incomplete. After validating, the expert could pick up where he left off. This is an error in the design phase on our part, as a better anticipation would have allowed us to have time to make this correction which we consider important.
To define which words are parsable and which are not, the use of an anti-dictionary would have been more effective than a list of separators to avoid the expert annotating words like "As" or "The".
We made sure to document the code precisely so that future developers can update the application and implement these interesting features as easily as possible.

3.3. Conclusion

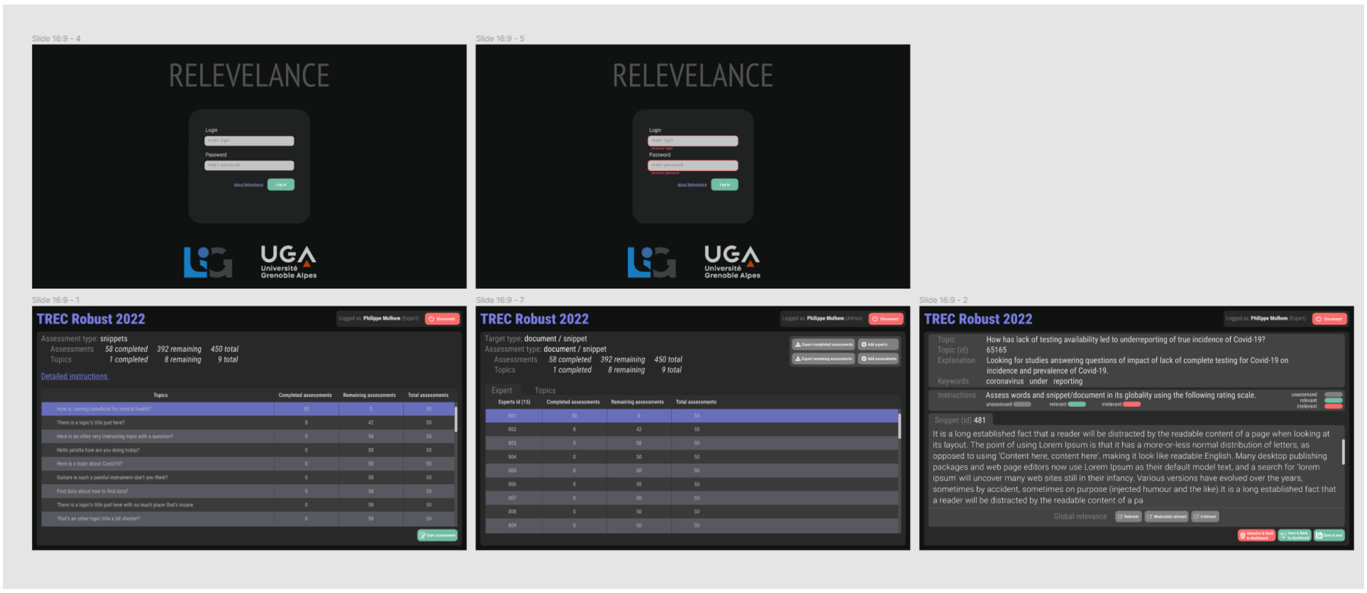
This project seemed to us both interesting for several reasons. First of all, it allowed us to get a foot in the world of computer science research by working in the computer science laboratory of Grenoble and by rubbing shoulders with researchers on a daily basis. It was interesting to observe the functioning of the teams and to be able to participate in formal and informal meetings. We are also satisfied with what the project taught us about project management. It was the first time that we had to design and develop such a technical application in its entirety from a specific request. We both feel that we have learnt a lot in this area.
Finally, we appreciated the possibility of applying all the theoretical knowledge we had obtained during the courses we took in our master and bachelor degrees. We needed our knowledge in software engineering during the design and needs analysis phase, in human-machine interface to design the interface, in server and client web programming for the development of the application (JavaScript, HTML, CSS, PHP, Laravel languages) and in information retrieval for the concepts of the application's use domain.
Overall this project was very positive for us and we are satisfied with the progress of the project and the result achieved in relation to the request.

4. Annexes

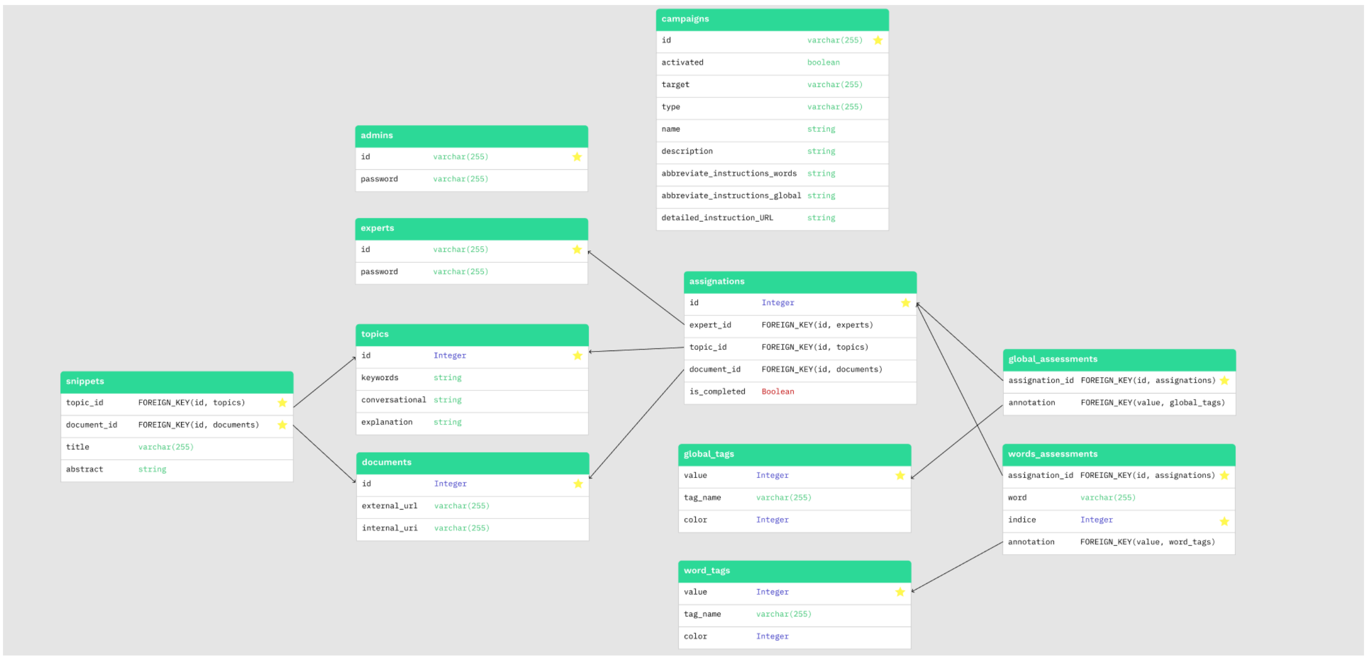


RELEVELANCE
Login
Password
RELEVELANCE
UGA Université Grenoble Alpes
TREC Robust 2022
Assessment type global/words
Assessments
Topics
Detailed instructions
TREC Robust 2022
Experts
Topics
TREC Robust 2022
Topic: how has lack of testing availability led to underreporting of true incidence of Covid-19?
Global relevance

Annex 1. Final version of the white theme mockup



Annex 2. Final version of the dark theme mockup (not implemented)



Annex 3. Schema of the database structure