



Ecole Polytechnique Fédérale de Lausanne

D-Voting : Front-end development for an e-voting platform

By : Ahmed ELALAMY, Ghita TAGEMOUATI, Khadija TAGEMOUATI

Supervised by :

Prof. Bryan Ford (DEDIS)
Advisor

Noémien Kocher (DEDIS)
Supervisor

Contents

1	Introduction	4
1.1	Abstract	4
1.2	Project Structure	4
1.3	Motivation	5
2	Designs and Implementations	7
2.1	Authorization mechanism	7
2.1.1	Context	7
2.1.2	Basic model	8
2.1.3	Casbin with an Adapter	11
2.2	Extension of Front-end functionalities	12
2.2.1	Context	12
2.2.2	Hint Button	12
2.2.3	Individual Results Display	15
2.3	Translation and Internationalization (i18n)	20
2.3.1	Context	20
2.3.2	Translation of the web page interface	20
2.3.3	Internationalization: Create a form	21
2.3.4	Internationalization : Ballot Display	23
2.3.5	Internationalization : Display the results	24
2.3.6	Mocks	25
2.4	Other changes	26
2.4.1	From election to form	26
2.4.2	Modify the introduction image	26
2.4.3	Fix cast vote button	26
2.4.4	Modify the JSON filename	27
2.4.5	MaxLength bug fix	27
2.4.6	Change the favicon	27
2.4.7	Add a contributors mention in the About page	28
2.4.8	Redirecting to the previous page when logging in	28
2.4.9	Add a confirmation before logging out	28
2.4.10	Fix setting a role and deleting a role to a user	29
2.4.11	Result Types	29
2.4.12	Edit a node	30

2.4.13	Ensure the verifiability of ballots in the front-end	31
3	Conclusion	32
3.1	Summary	32
3.2	User Testing and Feedback	32
3.3	Possible Improvements and Features to add	34
	Bibliography	36
A	Individual JSON export example	37
B	Translation	40

Chapter 1

Introduction

1.1 Abstract

To address the growing importance of e-voting in our lives, the DEDIS lab has been working with students to create new solutions. The D-Voting project [6], which is based on the Dela blockchain, promises to provide a system that guarantees voter anonymity while being completely auditable and decentralized. A web application was previously created in order to facilitate the communication with the blockchain itself. As a result, the objective of this project's fourth iteration is to enhance the front-end features that are already in place and add new ones.

1.2 Project Structure

Before going into the details of what has been done during the semester, let us first introduce the structure of the D-Voting project. A diagram can be found in Figure 1.1 below. The project has two main parts: The Blockchain part and the Web part, with the latter being our focus this semester. Within these two parts are four main "entities":

1. The **blockchain nodes** are the main focus of the Blockchain part, they manage all the voting logic of the project. They are built on top of the Dela blockchain with a smart contract specific to D-voting and two services: a Distributed Key Generator (DKG) and a Verifiable Shuffling service. They are numerous and communicate with each other with the gRPC protocol.
2. Each node contains a **proxy**, which is a HTTP server that makes it possible for a web server to communicate with the nodes in the blockchain. It is mainly called by the Web Back-End, but the Front-end also issues some requests.

3. In the Web Part, the **Web Back-End** is a web server handling authentication, via the available EPFL Tequila service, and authorization of the different users. It is trusted by the nodes, hence all the different voting requests are passed by the Back-End so that it can sign them. It is in constant communication with the Web Front-End.
4. The **Web Front-End** is the final and the most important entity of the project. It is a React-based Web App, styled with TailwindCSS [8], offering a view to the end-users of the system. Depending on the task, the Web Front-End will directly send HTTP requests to the proxy of a blockchain node, or to the Web Back-End.

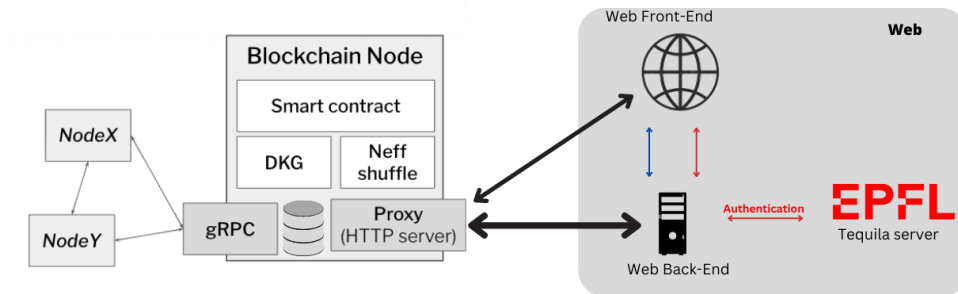


Figure 1.1: Structure of the D-Voting Project.

In the rest of this report, we will abbreviate Web Back-End into back-end and Web Front-end into front-end. Unless explicitly stated, when referring to filepaths, assume we are at the root of the project.

1.3 Motivation

Going into this project with no experience in web development, our personal goal was first and foremost to learn as much as possible on this topic that we would have never studied in any class.

The idea was first to implement possible improvements pointed out by previous students that worked on the project. Then, while working on that, we noticed some bugs and in collaboration with other students who are working on the project currently, we defined more possible features.

We focused on three main axes of improvement :

- Usability refers to the ease of use and learnability of a web application. It has good usability if it is easy for users to accomplish their tasks and achieve their goals without becoming frustrated or confused.

- Robustness refers to the ability of a web application to withstand and recover from errors, exceptions, and other unexpected events. This includes ensuring that the authorization mechanism is able to handle a wide range of inputs and conditions without failing or being bypassed, and that it is able to recover gracefully when errors do occur.
- Accessibility refers to the degree to which a web application can be used by people. This includes providing alternatives for users who are unable to use a mouse, or providing language selectors and allowing users to easily switch between languages, ensuring that translated content is accurate.

Chapter 2

Designs and Implementations

In this chapter we go into much details about what we have done during the semester and how we have done it.

2.1 Authorization mechanism

2.1.1 Context

Authorization mechanisms are a critical component of any system that controls access to resources. They help to ensure that only authorized users can access the resources they are allowed to, while protecting against unauthorized access. This is particularly important for protecting sensitive or confidential information, as unauthorized access to such resources could have serious consequences.

At the beginning of the semester, the authorization mechanism was as follow. When a user logged in, their information were retrieved. We used a user's database to store all users' information. If a user was in the database they got one of these three roles: Admin, Operator or Voter. If not, they had no rights. Only admins were able to modify the database.

While this system worked correctly, it had the limitation of only offering three roles. The first issue is that it can be difficult to determine the appropriate level of access for each role : Should an admin have all rights for all forms ? Also, the roles were not clearly determined. Indeed, admins and operators had the same rights except that operators could not control other users' roles. Another issue with this mechanism is that it can be difficult to maintain and update the system as the needs change.

2.1.2 Basic model

Design

When thinking about authorization mechanism, we think about access control. There are different ways to implement it, one of them is Capabilities. The idea is to store a matrix taking information in rows, i.e. associate the permissions to users. For each user, the Capabilities list contains a tuple for each object expressing the permissions for that object of the user.

To address the limitations we had earlier, we decided to implement an authorization mechanism using the Casbin library [2]. This system is based on the concept of policies, which are set of rules that defines the access control rules for a particular system or application. They allow us to grant specific permissions to users and then test whether a user has the necessary policy when it is required. These policies specify the actions that users, groups, or processes are allowed to perform, as well as the resources they are allowed to access.

We chose to use Casbin for several reasons :

- It uses a simple and expressive policy language that allows developers to define access control rules in a clear and concise manner. This makes it easy to understand and maintain the policies over time.
- It is really flexible.
- It is designed to be scalable and can handle large numbers of users and resources. This makes it suitable for use in high-traffic systems and applications.
- It includes a number of adapters that allow it to integrate with various storage back-ends.

Implementation

This is our first time using this mechanism, so we wanted to start with a simple approach.

To implement the new access control system, we started by creating a basic model. This will involve creating two files:

- One that outlines the model being used: `model.conf` (Figure 2.1.2 is used to define the access control model).
The `request_definition` section defines the variables that will be used to represent a request, to access a resource, or perform an action. The `policy_definition` section defines the variables that will be used to represent a policy, and the `policy_effect` section specifies the effect that should be applied when a request matches a policy. Finally, the `matchers` section defines the conditions that must be met in order for a request to match a policy.


```

[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act .

```

Figure 2.1: An example of a `model.conf` file.

```
p, "SCI", "subject", "action"
```

Figure 2.2: An example of a policy.

- the other file includes all the policies. It contains lines like the one in figure 2.2. The SCIPER is used to identify users.

But it was not just about creating these two files. The idea was to build on the previous mechanism and create policies for each action that had to check the user's role. We realized that some tests were done directly in the front-end. As, the authorizations are enforced on the back-end we needed to have information exchanged between the back-end and the front-end. Also, to avoid checking the policies by using Casbin each time we need it and to have access to user's rights from the front-end, we created a map that links each object to the action the user is authorized to perform and added it to the personal information of each user. This information is send to the front-end when logging in.

The behavior of the website is now as follows: when logging in, we get the user's personal information that contains their name, surname, SCIPER, role, a boolean islogged and authorizations. Then, according to the user's rights, the front-end adapts the view.

Using Casbin offers us a more flexible authorization mechanism than before and therefore lets us have more types of users. As we can see in figures 2.3, 2.4, and 2.5, we have a user that has all rights, one that doesn't have access to the Admin page but can still create a form. We can do whatever we want.

After making sure that all is working, a minor change is needed. With the new system, the role attribute for each user is no longer necessary. Therefore, we have decided to remove all references to roles.

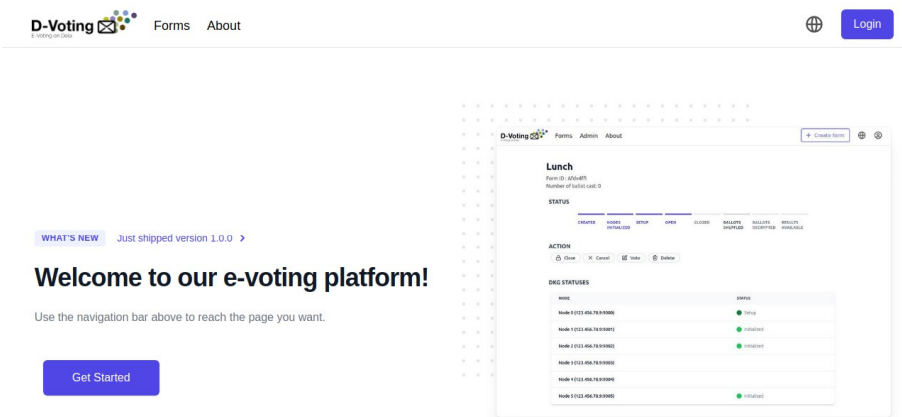


Figure 2.3: Before login, important pages cannot be accessed.

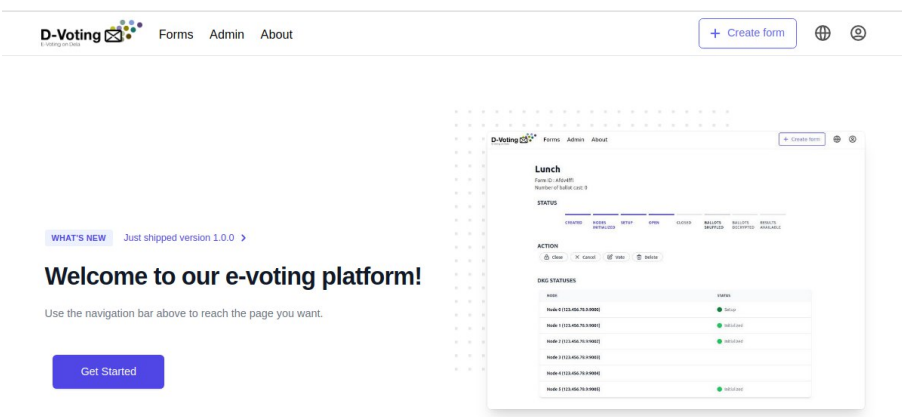


Figure 2.4: When a user has all access, all pages appear in the navigation bar.

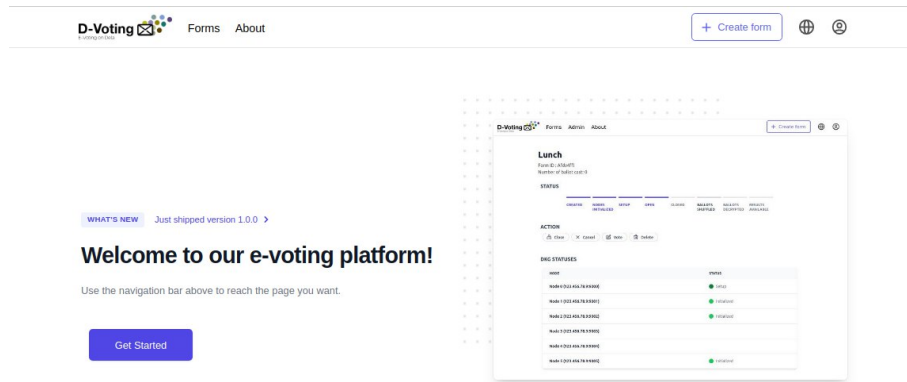


Figure 2.5: The user logged in cannot manage other users, so the Admin page doesn't appear.

2.1.3 Casbin with an Adapter

Design

For only eight users, the policy file had already more than sixty lines. Then, in the context of using this website at EPFL with a population greater than 15,000 people, it will not be a long term solution to store our policies in that file. To remedy this issue, as said before, Casbin offers adapters that allow it to integrate with various storage back-ends.

Implementation

After testing different databases services (mysql, postgres) and different adapters (sql-adapter, postgres-adapter, sequelize-adapter), we finally find something that works and that doesn't have an outdated source code. We use postgres [10] for our database and the sequelize-adapter [1] as our adapter. We chose to use Docker Compose [4] to define our database easily and allow us to start and stop all the services with a single command. We initialize our database in a file `migration.sql`, and then create the link between it and Casbin.

Finally, we thought it is more logical that someone who can create forms can only operate on the ones they have created. For that, we added a request from the front-end to the back-end so that each time someone creates a form, we add a new policy for the user involved using the formID to identify the form. For the other way around, we added an authorization test before doing some actions (initializing the nodes, shuffling the ballots, closing the form...).

Another change that we needed to do was to modify the mocks used when running the front-end without the back-end, so that we don't break it. We just needed to add the authorizations to the virtual user's personal information so that all works correctly.

id	ptype	v0	v1	v2	v3	v4
character varying (256)	character varying (256)	character varying (256)	character varying (256)	character varying (256)	character varying (256)	character varying (256)
1	p	330383	roles	list		
2	p	330383	roles	remove		
3	p	330383	roles	add		
4	p	330383	proxies	post		
5	p	330383	proxies	put		
6	p	330383	proxies	delete		
7	p	330383	election	create		
8	p	330382	roles	list		
9	p	330382	roles	remove		
10	p	330382	roles	add		
11	p	330382	proxies	post		
12	p	330382	proxies	put		
13	p	330382	proxies	delete		
14	p	330382	election	create		
15	p	228271	roles	list		
16	p	228271	roles	remove		
17	p	228271	roles	add		
18	p	228271	proxies	post		
19	p	228271	proxies	put		
20	p	228271	proxies	delete		

Figure 2.6: An extract from the database.

2.2 Extension of Front-end functionalities

2.2.1 Context

As it was at the beginning of the semester, the website was already functional and could do all the basic operations expected from a voting platform: Create, manage forms and participate in them. The main idea was to now focus on User Experience (UX) and add some features that are nowadays already implemented in most of the voting websites and that would make ours more enjoyable.

2.2.2 Hint Button

Design

When you answer a form online, there is generally a little clickable "(i)" icon that can give you more information about what is expected of you. The first example that comes to mind is when prompted to enter the CVV number of your credit card, for which you usually find some hint button explaining what it refers to and where to find it on your card.

What we call here a "Hint Button" is more generally called a "tooltip" and is a really useful component that can take multiple forms and is considered invaluable today by UX researchers [12]. A tooltip can be used in different scenarios [9]:

- Describe new features: For example, Twitter adds a new voice chat feature and when the user opens the app after updating it, a little popup right next to the new voice button appears to let them know about this new feature.
- Contextual help, such as the previous CVV example.
- Brief instructions: For example, you are answering an online survey that asks you to enter your address, a little information button is there to let you know what format is expected.

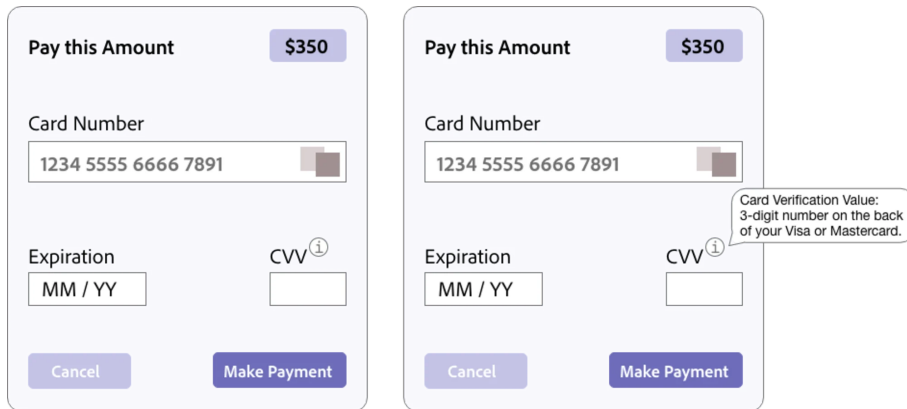


Figure 2.7: A web payment component with a hint for the CVV field.

The latter scenario being exactly what we will focus on in this implementation, as form creators may want to add more instructions and maybe even hints/tips to help the voter in their answers.

Implementation

Because adding a question-specific Hint field is by nature close to the structure of a question itself, this new feature implied many changes throughout the codebase.

In the D-Voting codebase, each type of Question (`Text`, `Rank`, `Select`) is defined separately in every part. As a question is entirely sent to the blockchain nodes at the creation of a form, it was necessary to add the specific `Hint` field to each of the question types definitions across the codebase. This included the front-end (`web/frontend/src`) type definitions in `types/configuration.ts` with their initializing functions in `types/getObjectType.ts` and the types in the smart contract in `contracts/evoting/types/ballots.go`. A little refactoring was needed because another thing was already called "Hint" in the codebase, and it was referring to the requirements for each question, e.g. for a `Select` Question the sentence that says "Select at most 3 answers" was considered a "Hint". To avoid any problems and confusion, these were refactored to "Requirement".

Changes were also necessary in the JSON validation. We deal with the JSON format for every communication with the back-end and when importing forms from a JSON file. In both these cases, the objects need to be correctly formatted. In `web/frontend/src/schema/configurationValidation.ts`, we have a schema and cross-field validation mechanism implemented using Yup [11]. In simpler terms, it is a function that checks that when we create a JSON object, there is no unnecessary fields and each field value respects some requirements

(e.g. the Max length for an answer in a Text question is at least of 1). At this point the `Hint` field would be added when the JSON object is created but it would not pass the validation as it is not specified in the validation schema. We therefore needed to update this as well by simply adding to the validation's configuration that there should be a string field named `Hint`, but with no further requirements as it should have the possibility of being empty.

In the visual aspect of the creation of a Form, we added an input field just like for the question title or the choices names, with the difference that we did not make it mandatory. Then, a new `/components/buttons/HintButton.tsx` component was added to the display of each question. This component has been placed next to the main title and has been styled to be a hoverable question mark icon. It has been mainly implemented using an already available "Popover" component by HeadlessUI [7]. The optional aspect was implemented by checking whether the text provided is empty or not and we create the button accordingly. Figures 2.8 and 2.9 show the Hint Button in action.

The image shows a modal window titled "Select" with a question mark icon. At the top, there are three language selection buttons: "German" (with a German flag), "English" (with an American flag and highlighted), and "French" (with a French flag). Below this, the form is organized into sections: "Main properties" containing "Title" (with a text input field containing "Enter the Title in English") and "Hint" (with a text input field containing "Enter a Hint in English (optional)"); "Additional properties" containing "Max number of choices" and "Min number of choices", both with dropdown menus set to "1"; and "Choices" (with a text input field containing "Choices 1" and a green plus icon). At the bottom, there are two buttons: "Cancel" and "Save" (with a checkmark icon).

Figure 2.8: The updated modal for a select question now providing a Hint field.

4
 5

How did you find the teaching ?

Select 1 answer.

bad
 normal
 good

Who were the two best TAs ?

Fill at least 1 answer.

TA1: 0 / 20

Be honest. This is anonymous anyway

Figure 2.9: The Hint Button in action.

The final needed change was in the mocks' data. As we designed the `Hint` field to be mandatory in the data structures and just empty if no hint was provided, the mocks' data needed to contain the field as well. We kept it empty for most questions, but also added a few examples in.

2.2.3 Individual Results Display

Design

When you have created a form online and you look at the results, there usually are two tabs where you can see the results in two different ways: Look at the averages on each question, or look at each Individual answer separately. An Individual answer, that we will also later call a ballot, refers to the choices selected and fields filled by a single user until they hit the "Submit" button. At the beginning of the semester, the D-voting project had the first way implemented, but throughout tests and discussions during the weekly meetings, we thought that displaying the Individual results could be a useful feature.

Assume for example that a form was created within the DEDIS lab to collect the pizza order of every lab member during a lunch break. Knowing that 50% of the lab ordered a pizza with pineapple on it is not a useful information, we would rather know who ordered what pizza with which drink. Another example we thought of is for correlation analysis: If your form contains two questions, for example "*From which faculty are you?*" and "*What is your favorite cafeteria?*", it is great to know that mostly IC people answered and that the preferred cafeteria across all answers is the Esplanade, but you might want to know if being in SV and liking L'Ornithorynque is correlated.

Implementation

At the beginning of the semester we had a single `Result` page in `web/frontend/src/pages/form/Result.tsx`. Because we wanted to have different types

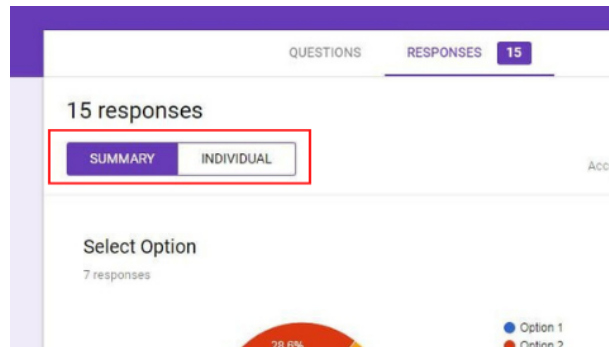


Figure 2.10: An example in Google Forms of two different tabs for what they call "Summary" and "Individual" results.

of display, we decided to create two files that would be referenced in the **Result** page: `web/frontend/src/pages/form/GroupedResult.tsx` that would generate the same results as what we had before, and `web/frontend/src/pages/form/IndividualResult.tsx` which would generate the separated results for each ballot.

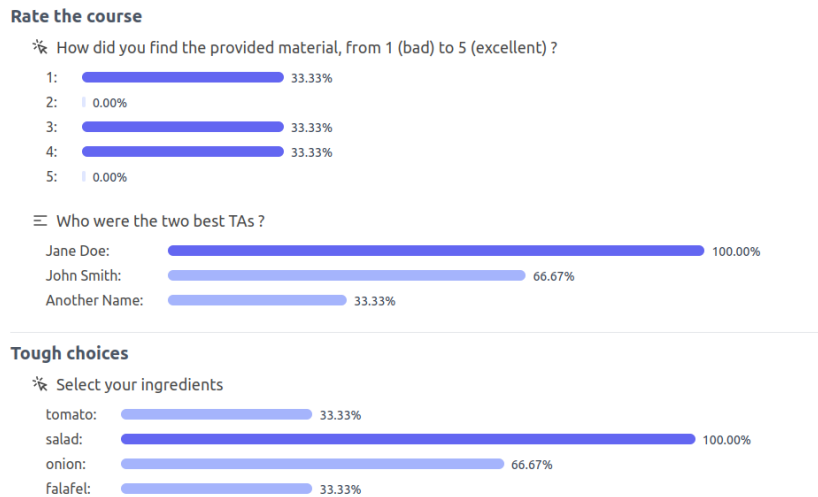


Figure 2.11: The Grouped Result display of a form (unchanged).

The first thing that we needed to do was divide the two types of display in two different tabs. This was easily possible thanks to the **Tabs** component by HeadlessUI. Each tab is called "Grouped" and "Individual" and creates the according type of display.

Results

Total number of votes : 3

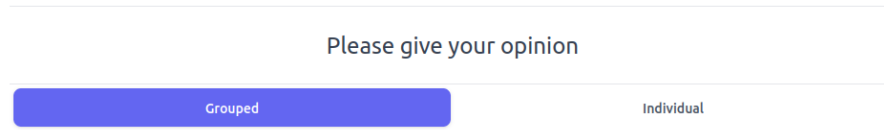


Figure 2.12: The two result tabs.

A useful property that we already had at the beginning of the semester is that, when we retrieve the results from the blockchain, we actually just retrieve an array of the ballots that were sent, with the sole difference that they have been shuffled and anonymized. It is actually the front-end that is responsible of averaging the results in the file `web/frontend/src/pages/form/components/utlis/countResult.ts`. So what we needed to do was to choose which ballot to display, and then pick it within the provided array.

We went into this with the idea of reusing as much code as possible. Therefore, we first reused the same display components (`TextResult`, `SelectResult`, `RankResult`). These components take as parameter a two-dimensional array, i.e. an array of each ballot answer for that question, which is itself an array. As here we are only taking one ballot, we just need to take the 1D array and wrap it back into a 2D array. All the methods that would count and average the results could still be used as they all expect a 2D array and averaging a single answer will just return that answer.

This is where the first version ended, but we quickly realized that keeping the same type of display was not enjoyable at all. For example, this would show the `RankResult` as percentages (the 1st place at 100%, and so on...), when it would make more sense to put the 1st place first, then the 2nd place second, and so on.... Therefore, we needed to redesign those graphic components to make them more comprehensible. For each type of result, we added an individual result type as well, and each does the following (shown in figure 2.13):

- `IndividualTextResult` shows the name of the field in bold and the answer given for that field.
- `IndividualSelectResult` shows each choice with a box next to it, which is checked if the answer was chosen.
- `IndividualRankResult` shows the choices in the order they were ranked.

Now that we have prettier individual results, we needed a way to navigate through them, i.e. a pagination mechanism. The first thing we put in place is a

Rate the course

✳ How did you find the provided material, from 1 (bad) to 5 (excellent) ?

- 1
- 2
- 3
- 4
- 5

☰ Who were the two best TAs ?

TA1: Jane Doe

TA2: John Smith

Tough choices

✳ Select your ingredients

- tomato
- salad
- onion
- falafel

Figure 2.13: The Individual Result display of the first Ballot in a form.

stateful variable within the `IndividualResult` that keep in mind which ballot we look at currently and refreshes the display each time the ballot selected is changed. We first only added "Previous" and "Next" buttons, which worked well but were not useful when we would have more than a few ballots. We needed to add a way to enter manually the number of the ballot we want to look at. In Google Forms for example, this is done using a number-type input field, but after doing some research, we realized this would potentially cause problems [13]. We therefore decided to go with a text-type input field with our own verification procedures. If what was inputted is not an integer and/or is not within the range, then an error message appears and the ballot is not updated. This is shown in figure 2.14.

The last thing that needed to be updated is the JSON export. There is a possibility to get the results in a JSON format. With the grouped results this would go question by question and for each choice show the percentages. It seemed logical to export the results individually as well, but with the same logic that there might be a lot of ballots, we decided to put all the ballots in the JSON file, not only the one being currently displayed. Also, as here we have no percentages, the name of each field in the JSON file made no sense anymore. Therefore, besides adding a Ballot Number at the beginning of each ballot being

Results

Total number of votes : 3

Please give your opinion

Grouped Individual

Previous Next

Results

Total number of votes : 3

Please give your opinion

Grouped Individual

Previous Next

Please enter a number between 1 and 3.

Results

Total number of votes : 3

Please give your opinion

Grouped Individual

Previous Next

Please enter a number between 1 and 3.

Figure 2.14: The Pagination mechanism working and displaying errors.

displayed, we changed the names of each type of field:

- For Text results we chose to put the **Field** name and the **Answer** provided
- For Select results we put each **Candidate** with a **Checked** value that is either true or false.
- For Rank results we put each **Rank** in order and which **Choice** was placed at that rank.

Lastly, the names of each export were suffixed by ”_grouped” or ”_individual” accordingly. Appendix A shows an example of an Individual Result JSON compared to a Grouped Result one on a Rank question.

2.3 Translation and Internationalization (i18n)

2.3.1 Context

Internationalization and translation are important considerations for any software project that aims to reach a global audience. These processes involve adapting the software to be able to support multiple languages and cultural differences, allowing users from different linguistic backgrounds to use the software in their preferred language. There are several reasons why internationalization and translation are important for a software project but the most important for us is that it helps to improve the user experience for non-native speakers. This can lead to higher levels of satisfaction and engagement among users.

This project is meant to be used by the EPFL community. Therefore, at the request of EPFL, we first wanted to translate the interface into the three languages used: English, French and German.

2.3.2 Translation of the web page interface

The translation of the interface was already implemented on the project. The only thing that was needed to be done is to fill in the `fr.json` and `de.json` files. But, first, let us explain how it works.

Design

There is a folder `web/frontend/src/language`. On that package we have three files: `en.json`, `fr.json` and `de.json`. These files contain all the translations that we need for the interface. We have two other files: `LanguageSelector.tsx`, which is the component that is used to change the language displayed, and `configuration.ts` configures the language using the packages of internationalization provided by React. Finally, we used a function `useTranslation` imported from `i18next` [5] to use one of the 3 files based on the current language.

Implementation

To complete the task, we needed to fill in the `fr.json` and `de.json` files. To do so, we used DeepL [3], a reliable translation service. However, none of us speak German, so we asked friends and colleagues to check the translation for accuracy. Additionally, we realized that some features had not been translated because they were not included in the JSON files. We made an effort to add all displayed features to these files as much as possible. Appendix B shows an example of a page in the three languages.

2.3.3 Internationalization: Create a form

Now that we have implemented the translation on the web page, the next step is to enable users to select the language they prefer for the forms. To ensure accurate translation of the forms, which are entered by other users, we have decided to allow the option of adding translations in the three languages when of creating the form.

Design

First let's explain how a form was created. The elements of a form are defined as a configuration, which consists of a main title and a scaffold. The main title is the overall title of the form, and the scaffold is an array of subjects that are being voted on. There are four types of subject elements, each of which has approximately the same features. The ones that we are interested in are the subject title, the list of choices, and the hint button. Once the form is created, all of this information is sent to the back-end for processing.

Implementation

We focused on the front-end of the system. As previously mentioned, we want to be able to display the form in three different languages. To do this, we started by working on the main title of the form. Initially, we created three fields for each language and saved the values in separate variables: `MainTitle`, `TitleFr` and `TitleDe`. We added these variables to the configuration so that we can display the correct value for the main title in each language. We also added separate variables for the subject titles. At this time of the project, we had three boxes for each languages where we can enter the `MainTitle` and three other boxes for the Subject Title.

Next, we wanted to be able to enter the choices for each subject in three languages. Previously, the choices were stored in an array. However, we realized that adding two additional fields for each choice would be too cumbersome and make the form difficult to create, especially since there could be a large number of choices. For example, if a subject initially had five choices, our method would require fifteen fields to be filled out in order to enter all the values in three languages. To solve this problem, we added three buttons at the top of the pop-up window that appears when adding a subject (Figure 2.15). These buttons allow the user to choose the language and enter the values for that language. The choices are initially stored in three separate arrays: `Choices`, `ChoicesFr` and `ChoicesDe`.

For consistency, we did the same for the main title and the subject title (Figure 2.16).

Now we can enter and store all the elements that we want but we did not deal with the back-end yet. Our goal was not to change the back-end. We therefore

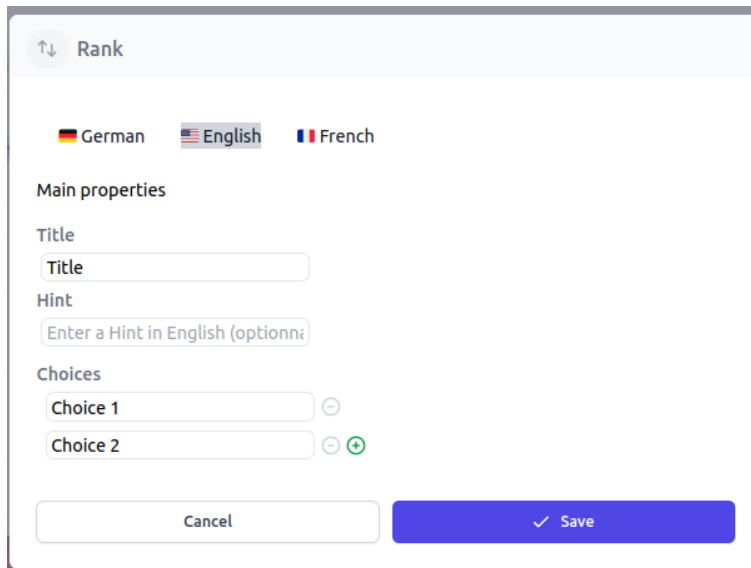


Figure 2.15: The updated modal for a rank question now providing 3 buttons for the language.

decided to stringify the three titles into one string according to the following scheme:

```
{'en' : 'MainTitle', 'fr' : 'TitleFr', 'de' : 'TitleDe'}
```

Once again, we did the same thing for the subject titles and the hint.

The choices presented more complexity: Our initial plan was to change the way we stored the choices by using a map that associates a language with an array of values for that language. However, implementing this change proved to be difficult because maps in TypeScript often cause problems. In fact, we lost a lot of time handling errors caused by the maps. Eventually, we realized that the back-end cannot handle a map and expects to receive an array instead, which is why we thought of a way to change our map INTO an array. We added a new variable named **ChoicesMap** which will be our map and **Choices** will stay an array and be sent to the back-end. All together, this is how it works:

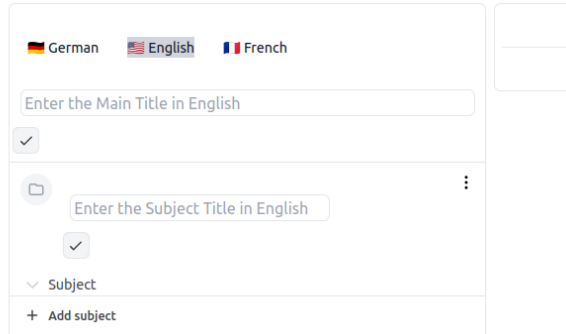
When we create a subject **ChoicesMap** is filled, but when we want to change the elements before going to the back-end we call a method called **ChoicesMapToChoices** that returns an array containing all elements of the map according to the following scheme:

```
[{'en' : 'Choice 1', 'fr' : 'Choix 1', 'de' : 'Auswahl 1'}, {'en' : 'Choice 2', 'fr': 'Choix 1', 'de': 'Auswahl 2'}]
```

Now all the elements of the configuration are well stored and sent to the back-end.

Create

Create a new form by filling out the information below or by [uploading a JSON file](#)



The screenshot shows a form titled "Create" with a subtitle "Create a new form by filling out the information below or by [uploading a JSON file](#)". At the top, there are three language selection buttons: "German" (with a German flag), "English" (with an American flag and highlighted), and "French" (with a French flag). Below the language buttons is a text input field labeled "Enter the Main Title in English". Underneath this is a checked checkbox. The next section is a "Subject" section, which includes a text input field labeled "Enter the Subject Title in English", another checked checkbox, and a vertical ellipsis menu icon. Below the subject input is a "Subject" section with a dropdown arrow and the text "Subject". At the bottom of the form is a button labeled "+ Add subject".

Figure 2.16: The updated create form page with the 3 buttons for the language.

2.3.4 Internationalization : Ballot Display

The `BallotDisplay` component is used to show the ballot but also to display a preview when the user is creating a form.

Design

First, we want to be able to display the ballot correctly according to the language chosen. Previously, to display the elements, the form was being taken from the back-end and all was displayed in English. We will need to convert all the elements of a form that we added to the new configuration on the front-end as expected by the back-end

Implementation

As we stringify the title, to be able to display the right value, we parse our `MainTitle` element and we assign the English value to `MainTitle`, the French value to `TitleFr` and the German value to `TitleDe`. We do this in the `BallotDisplay` file.

For the hint and the subject title it is the same procedure but we did it on the `Rank`, `Select`, and `Text` files. Finally, using `i18next` we can use the language function and display the right value depending on what it returns.

For the choices, once again it is more complex. As we already explained, when creating a form we add our choices to a map and we change this map to an array before sending it to the back-end. So to be able to display the right value we go the other way around. That means we define a function `choicesToChoicesMap`, that takes the array sent by the back-end and fills the `ChoicesMap`. Then, depending on the language chosen by the user, the right values will be displayed. This is done on the `Rank`, `Select` and `Text` files too (Figure 2.17).

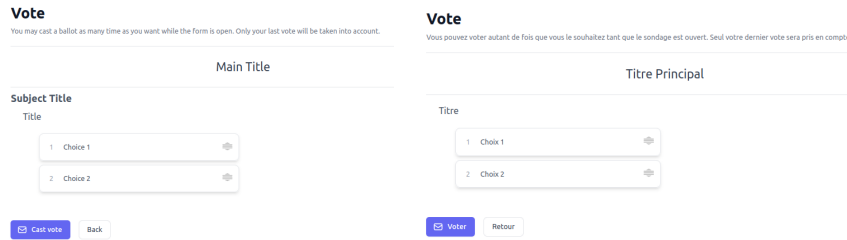


Figure 2.17: An example of the ballots displayed in English and in French.

Next, we want to be able to display the preview when the user is writing the form. After a lot of reflection, we thought that it is better and more logical for the user that the language displayed is the language that they are currently filling. We therefore added the language as an argument to the `BallotDisplay` component so we can use it in two ways:

- When we want to display the ballot to be able to vote, the language passed as argument will be the current language on the page.
- When we want to fill the form during the creation, the language passed as argument will be chosen according to the buttons that we added.

2.3.5 Internationalization : Display the results

Design

As for the ballot display, we want to be able to display the results in the right language. There was one component that was used to display the results : `Result`. We then added `IndividualResult` and `GroupedResult` in the mix. The main title and the subject title were handled in these files and the display of the subject Rank, Select and Text were handled in the `RankResult`, `SelectResult` and `TextResult` components. Now we need to fix these files in the way that it can display the correct languages

Implementation

We needed to change the display of the `MainTitle` for the `Grouped` and `Individual` results. First, as for the ballot display we needed to parse our `Title` into three elements. Using `i18next` we display the right value depending on the language. Once again, we did the same for the subject titles.

As for the choices, it is more complicated. Currently, we simply display an array that is mapped to the correct translation. (Figure 2.18)



Figure 2.18: An example of the display of the grouped result in English and French.

2.3.6 Mocks

Finally, we needed to change the mocks so that they are consistent. Now the values of the main titles, subject titles and choices are returned to string or an array of string following the schemes that we explained above.

2.4 Other changes

2.4.1 From election to form

The project was originally intended to be an e-voting system for EPFL elections. However, as it has grown and as the way in which EPFL intends to use it has evolved, we have decided to change the focus from elections to forms. This is why we have modified the use of the word "election" to the word "form".

2.4.2 Modify the introduction image

The website has undergone some changes, and as a result, the introduction image on the homepage (which was a screenshot of the form page) was no longer accurate. We have replaced the image with a new one to reflect the current state of the website.

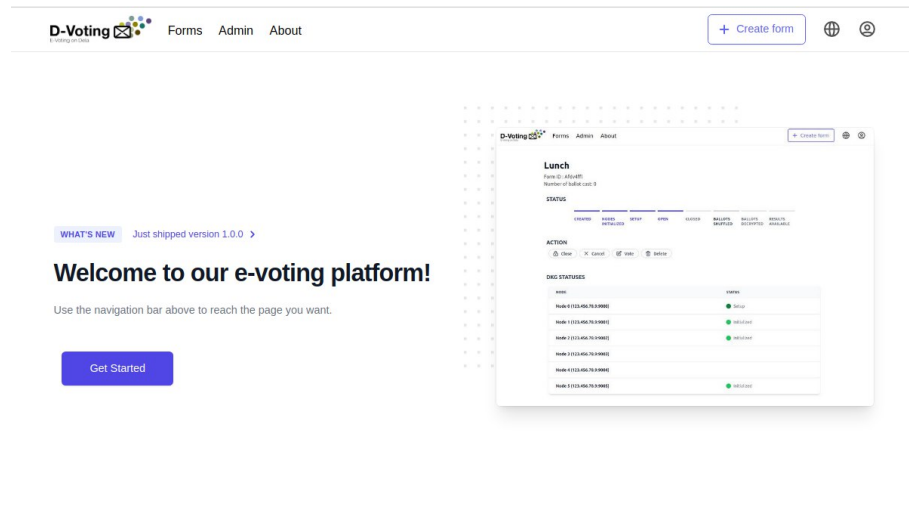


Figure 2.19: The new home page with the use of 'form' and the replaced image.

2.4.3 Fix cast vote button

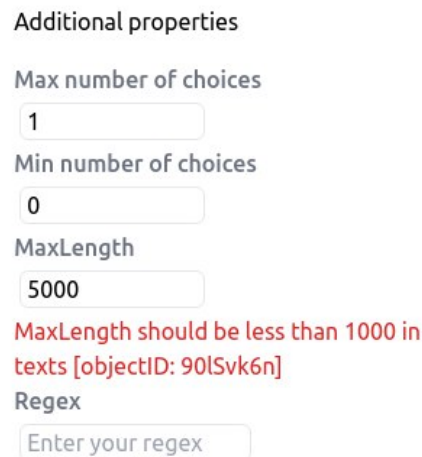
While testing the website, we noticed that it takes a little bit of time to load when you click the button to cast your vote. During this loading time, we tried clicking the button again. However, we noticed that this caused an unexpected behavior: we were able to cast multiple votes. To fix this, we have implemented a feature to disable the button while it is loading.

2.4.4 Modify the JSON filename

One useful feature of the project is the ability to create a form either through the interface or by uploading a JSON file. Similarly, when you have finished creating a form, you can download the JSON file. During testing, we realized that naming all files "form_configuration.json" was not logical. To improve this, we have changed the file naming to use the form's name instead. In order to avoid potential security issues, we have also decided to only use the first 100 characters of the title and to replace any non-alphabetic or non-numeric characters with underscores '_'. The same was done for the export of form results.

2.4.5 MaxLength bug fix

When creating a text question, the form administrator can specify a regex and a maximum length. However, the administrator can set the maximum length to any number greater than or equal to zero, which could potentially create a security issue if the text is very long. To address this, we have decided to restrict the maximum length to a value between 1 and 1000, inclusive. This will prevent administrators from setting the maximum length to a value that is too high.



The screenshot shows a form configuration interface with the following elements:

- Additional properties** (Section Header)
- Max number of choices** (Label) with an input field containing the value **1**.
- Min number of choices** (Label) with an input field containing the value **0**.
- MaxLength** (Label) with an input field containing the value **5000**.
- Error Message:** "MaxLength should be less than 1000 in texts [objectID: 90ISvk6n]" displayed in red text below the MaxLength input.
- Regex** (Label) with an input field containing the placeholder text "Enter your regex".

Figure 2.20: Here, we can see that an error appear when the maxLength is set to 5000

2.4.6 Change the favicon

A minor visual implementation was to change the favicon, which is the icon appearing next to the page name in a browser tab. On a React App, it is by default the React icon, but now it has been changed to the D-Voting Logo.

2.4.7 Add a contributors mention in the About page

The About page of the website is one of the pages accessible by any end-user. It contains some general information about the project, but as it is where interested people will go to learn more about the behind-the-scenes, it was logical to add a paragraph mentioning the names of all people having contributed to the project, just like it is available on GitHub.

2.4.8 Redirecting to the previous page when logging in

When logging in, the system would redirect the user to the home page of the website. This felt inconvenient and not logical in some cases. For example, when trying to access the admin page of a form without being logged in, the website would prompt you to login in order to determine whether or not you are able to see the admin info of the form. If you were to log in, the most logical thing would be to redirect you to that form page so that if you have access you can see the content and if not you get an error, but going back to the home page is just cumbersome.

To change that, we needed a way to keep the state of where the user was on the website when clicking on the Login button, so that we could go back there when we would return from Tequila. Keeping global state, track and save information about each user's session can easily be done by using HTTP cookies. Now, when clicking on the Login button, the corresponding handler will add a cookie that contains the path needed to redirect and that will expire after 2 minutes which should be enough time to build a connection. When the Tequila server answers and we are able to log in the user, the front-end then gets the cookie and navigates back to the address contained in it. If the cookie is not found because of an issue (Expiration for example), then by default the home page will be accessed instead.

2.4.9 Add a confirmation before logging out

At the beginning of the semester, we noticed that when a user try to log out they can do it too easily. But, if they log out while filling a form, then everything done would be lost.

To avoid this behavior, we wanted to be able to add a confirmation before logout. This way, the user will be aware that logging out will make them loose everything done so far. To do this, we first create a new component: a warning modal. This component is designed so that it can be used later for other purposes. It has been mainly implemented using an already available "Dialog" component by HeadlessUI. It takes 4 arguments:

- `isShown` : is true when the pop up is displayed
- `setIsShown` : a function that takes a boolean and sets `isShown` to true when the user click on the logout button

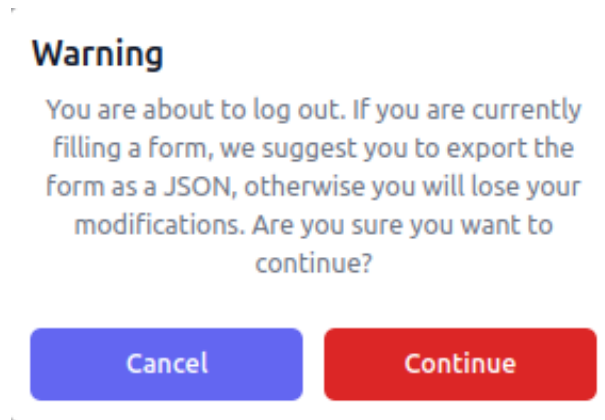


Figure 2.21: The warning modal when trying to log out.

- **action** : the action taken when clicking on the "continue" button. Here the action is logout.
- **message** : the message that is displayed and that warns the user of the behavior caused by the action and propose them to export the form if needed.

2.4.10 Fix setting a role and deleting a role to a user

When wanting to add a role or delete one, the behavior was unexpected. We needed to refresh the page so that the user is added or deleted.

To fix this, we used the same mechanism as for the proxy editing: instead of using the `fetch` function to make the right request, we create a function `sendFetchRequest` which calls the component `usePostCall` in which we handle differently the request.

Also, we realized that we had a problem deleting the last user. And to change that, we made a minor change on the `AdminTable` file. In fact, at the beginning when users change there was a `useEffect` that called `setScipersToDisplay` when the length of the users array is not null, but it didn't handle the case when the length was null so we added it.

2.4.11 Result Types

To make it even easier to understand, each question is now displayed with the Icon corresponding to its type next to its Title.

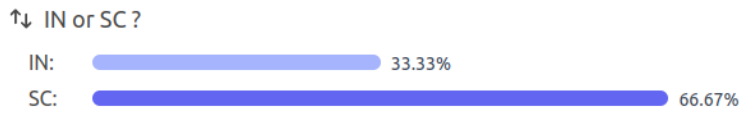


Figure 2.22: An example of a Rank question result with the Rank Icon next to the title.

2.4.12 Edit a node

On the admin page, if you have the appropriate permissions, you can manage nodes. However, when trying to edit an existing node, you can only edit the proxy mapping to it, not the node itself. We are providing a way to modify that as well.

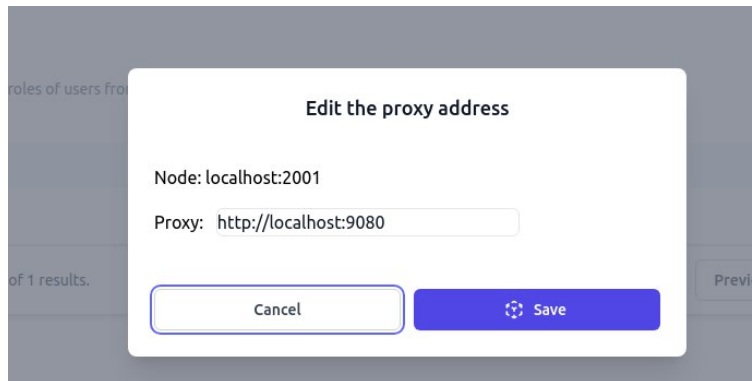


Figure 2.23: Before we could only modify the proxy address.

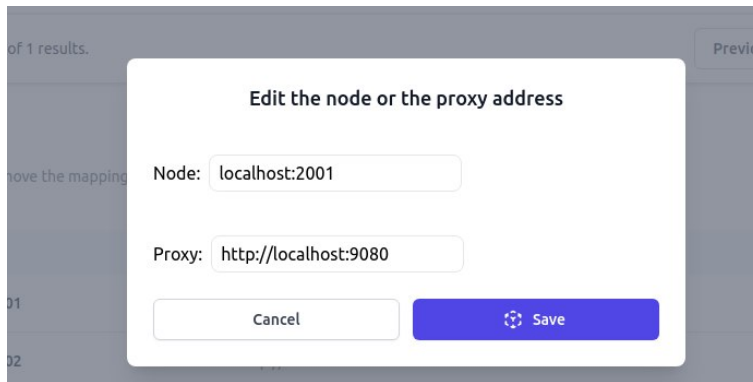


Figure 2.24: Now we can modify the node itself and the proxy address.

2.4.13 Ensure the verifiability of ballots in the front-end

Something that came up during the security audit was that there is no verifiability process in place for the ballots. Verifiability refers to the ability of an end-user to check whether their actions were taken into consideration or not. In the project, we could already verify whether a form that we created was successfully transmitted in the blockchain. When sending the form, the front-end would show the form creator a hash of its form, that could then be used to check in the blockchain if a transaction with the same hash is present.

To add this for the ballots we decided to put in place the same mechanism: when sending the ballot to the back-end, the front-end shows a popup with the hash of the ballot.

Chapter 3

Conclusion

3.1 Summary

Most changes that we managed to do were in order to improve usability. This made the most sense with the added intention of launching the application as soon as possible. We were still able to ensure more accessibility by answering EPFL's request to internationalize the system such that it is available in the school's three main languages. We improved robustness by improving the authorization mechanism with Casbin and correcting unwanted behaviors.

3.2 User Testing and Feedback

Throughout the semester, to have an outside perspective, we continuously asked colleagues for feedback. At the end of the semester, we asked a few friends, family members and colleagues to test the website with all the new features we added. The idea behind this testing was to let them discover the website, prompting them to do some essential tasks like vote in a form, create their own form, but still letting them use the features freely. As they tested, we asked them for feedback, here are the most important parts :

- A Data Science student pointed out that the export feature for the results was very useful, but that treating data in a JSON format might not be easy. He therefore suggested to add the possibility to download the results in a different format such as CSV.
- A student found that it might be useful to sort the forms in some other ways than by status. For example, maybe sort the forms to only show the ones in which the user did not vote yet, or to only display the forms they created.
- When creating a form, the user was uncertain about the purpose of the various titles. They also struggled to understand how to add a new question. To address this, the user suggested adding a button labeled 'add a

question' in the top right corner to make the process more clear. Additionally, the names of the different types of questions were not immediately understandable.

- The three languages within the website were well received, however a little inconvenience is that we are not able to see which language we are currently in. Multiple users suggested to display the flag of the current language on the website.
- During the form creation, a user suggested a different approach to fill the three languages: Instead of having buttons controlling all fields, we could maybe use a dropdown showing the translation already filled in and the ones still empty.
- The user accidentally clicked the button to create the form before completing it, and was disappointed to discover that they could not make any changes to the form after it was created.
- When creating a form, a user received an error saying its form was incomplete and could therefore not be created, but as they added many questions and subjects, they were confused as to where the error came from. A better error display could have helped highlight the locations where the form was incomplete.
- Overall, the users really appreciated that they can enter the 3 languages and it seems really instinctive to have the language that we are filling displayed.
- Some users found it very interesting to have a grouped result and an individual one. It was quite intuitive for them and they thought it could be very useful for some cases.
- A computer scientist pointed out to us that creating a form can be difficult for someone that doesn't have any knowledge on blockchains, as there isn't explanation as to what the system is doing at each step.

3.3 Possible Improvements and Features to add

Although we are really happy with the results obtained after this semester, we still have thought of possible ways in which we could make the Web App better. We will try to work on some of these improvements until the presentation because some of them seem quite important.

- We started this project without any experience in web development, we therefore spent our summer learning about JavaScript, Typescript, and React. It took us a lot of time to understand the relationship between the files and how the features worked because of the lack of the types of elements. Therefore, the first step to improve the programmers' experience is to try to use the strict mode of TypeScript as much as possible. We also noticed a lack of documentation for the web part. It is very difficult to understand how the system works just by looking at the code without any explanations. Therefore, the second step is to add some documentation to explain in details how authorization works, how a form is created, and finally how the ballots and results are displayed.
- Right now, the individual results can be seen by anyone. A possible enhancement would be to make them only available to admins just like in other form tools.
- As mentioned before, the new authorization mechanism does not require the role feature, so we deleted all references to it. However, we did not have time to update the admin page that displays a list of users with their roles.
- For now, we need to add manually user's authorizations except the ones for forms. It would be useful to have a feature that allows us to add users and assign them a set of permissions directly on the website. We can do something similar of what was done for managing users roles but instead of giving a role we can give a set of authorizations.
- Improve the way we can create a form in the 3 languages. Although we did our best to finish this feature we realize that there still some improvements to add. Right now, someone can enter partially the element of a language and only these elements will be displayed. So for example if you only add the title in french it will be the only thing displayed. There is two possibilities to improve this, either make it impossible to only enter partially a language information by changing the `configuration_validation` file or displaying the English values if not all the information is available. Also, we still have some bugs that break the shuffling when we vote using different languages. Finally, it seems like importing a JSON file to continue creating a form is not working too. We have to be able to parse the elements before uploading them

- Modularize the code, especially for the languages part. In fact, it can be a possible improvement to allow the user to enter it's own language or simply to be able to add a new language.
- Find a way to make a "draft" of the forms so that upon logging out, forms that are not finished yet could be resumed later. This is already possible via the JSON export, but other possibilities could be explored.
- As explained before, we had three roles to identify users. The Admin and the Operator are perfectly reproduced by our new system. However, the third role that is Voter, was used in the sense that not all people can vote. This gives an issue that someone needs to decide who can and cannot vote. This can be done by users with the right to manage other users. Furthermore, we noticed that one may want that only a group of users can vote for its form. This new behavior was not supported but it could be good to have it. The idea is to extend the actual authorization mechanism by using a new model, like a group based access control so that we can give the right of voting to a group for example.
- As pointed out by the security audit, we could also show the hash of its ballot on the form's Admin page, but this could potentially result in a cumbersome display.
- Find a better way to ensure verifiability. As ballots or forms can be long, their hash most of the time take more place than a little "Success!" popover. Therefore their display is not guaranteed to be elegant and convenient. This was planned to be investigated this semester but some more pressing enhancements came in the meantime.

Bibliography

- [1] Node Casbin. “Sequelize Adapter”. In: (Dec. 2022). URL: <https://github.com/node-casbin/sequelize-adapter>.
- [2] Casbin. “Casbin”. In: (Jan. 2023). URL: <https://github.com/casbin/casbin>.
- [3] DeepL. “DeepL Translator”. In: (2019). URL: <https://www.deepl.com/translator>.
- [4] Docker. “docker/compose”. In: (May 2021). URL: <https://github.com/docker/compose>.
- [5] i18next. “react-i18next”. In: (Jan. 2023). URL: <https://github.com/i18next/react-i18next>.
- [6] DEDIS Lab. “D-Voting”. In: (Dec. 2022). URL: <https://github.com/dedis/d-voting>.
- [7] Tailwind Labs. “tailwindlabs/headlessui”. In: (Dec. 2022). URL: <https://github.com/tailwindlabs/headlessui>.
- [8] Tailwind Labs. “tailwindlabs/tailwindcss”. In: (Oct. 2022). URL: <https://github.com/tailwindlabs/tailwindcss>.
- [9] Eric Olive. “Designing Better Tooltips For Mobile User Interfaces”. In: (Feb. 2021). URL: <https://www.smashingmagazine.com/2021/02/designing-tooltips-mobile-user-interfaces/>.
- [10] PostgreSQL. “postgres/postgres”. In: (Oct. 2019). URL: <https://github.com/postgres/postgres>.
- [11] Jason Quense. “jquense/yup”. In: (July 2020). URL: <https://github.com/jquense/yup>.
- [12] Sofia Quintero. “Tooltips: your secret weapon for improving feature discovery”. In: (Dec. 2018). URL: <https://uxdesign.cc/tooltips-your-secret-weapon-for-improving-feature-discovery-e1c380562f2e>.
- [13] Jared Toporek. “Why the number input is the worst input”. In: (Sept. 2022). URL: <https://stackoverflow.blog/2022/09/15/why-the-number-input-is-the-worst-input/>.

Appendix A

Individual JSON export example

Please give your opinion

Tough Choices

Which cafeteria serves the best coffee ?





1	Esplanade	
2	Giacometti	
3	Arcadie	
4	Montreux Jazz Cafe	

Figure A.1: An example of question.

```

1      {
2      "Title": "Please give your opinion",
3      "NumberOfVotes": 3,
4      "Results": [
5      {
6
7          "Title": "Tough choices"
8      },
9      {
10         "Title": "Which cafeteria serves the best coffee ?",
11         "Results": [
12         {
13             "Candidate": "Esplanade",
14             "Percentage": "55.56%"
15         },
16         {
17             "Candidate": "Giacometti",
18             "Percentage": "83.33%"
19         },
20         {
21             "Candidate": "Arcadie",
22             "Percentage": "83.33%"
23         },
24         {
25             "Candidate": "Montreux Jazz Cafe",
26             "Percentage": "77.78%"
27         }
28     ]
29 }
30 ]
31 }

```

Listing 1: The grouped JSON export on the example question.

```

1  {
2    "Title": "Please give your opinion",
3    "NumberOfVotes": 3,
4    "Ballots": [
5      {
6        "BallotNumber": 1,
7        "Results": [
8          {
9            "Title": "Tough choices"
10         },
11         {
12           "Title": "Which cafeteria serves the best coffee ?",
13           "Results": [
14             {
15               "Rank": "1",
16               "Choice": "Montreux Jazz Cafe"
17             },
18             {
19               "Rank": "2",
20               "Choice": "Arcadie"
21             },
22             {
23               "Rank": "3",
24               "Choice": "Esplanade"
25             },
26             {
27               "Rank": "4",
28               "Choice": "Giacometti"
29             }
30           ]
31         }
32       ]
33     },
34     {
35       "BallotNumber": 2,
36       "Results": [
37         {
38           "Title": "Tough choices"
39         },
40         {
41           "Title": "Which cafeteria serves the best coffee ?",
42           "Results": [
43             ...

```

Listing 2: An extract of the individual JSON export on the example question.

Appendix B

Translation

D-Voting Forms Admin About + Create form

ABOUT THE PLATFORM

What makes us different

The following diagram pictures the d-voting system from a deployment point of view. It describes the components and their interactions.

This website hosts the interface of an evoting system. This system runs smart contracts, handled by a set of Byzantine fault-tolerant nodes.

When an administrator creates a form, the form parameters are saved on a blockchain and so are every following transaction (closing/cancelling form, casting a vote etc...).

A distributed key is generated at form creation time so that when a user votes, his/her vote is encrypted with the key guarantying the anonymity of the vote. However the system doesn't enforce the anonymity of the voter.

When a form is closed, the nodes shuffle the ballots and check its correctness before decrypting the shuffle and publish the result of the form on a smart contract.

```
graph TD
    Admin[Admin] -->|Create form| Blockchain[Blockchain]
    Blockchain -->|Saved on Blockchain| Admin
    Blockchain -->|Distributed Key| Nodes[Nodes]
    Nodes -->|Merge election and shuffle| Blockchain
    Blockchain -->|Decryption| Nodes
    Nodes -->|Publish result on Blockchain| Blockchain
    Admin -->|Web Browser (Admin) ->|Close/Cancel form| Blockchain
    Admin -->|Web Browser (Admin) ->|Get voting info from blockchain| Blockchain
    User[User] -->|Web Browser (User) ->|Cast vote| Blockchain
    Blockchain -->|Encrypted with Key| User
```

Figure B.1: The about page in English.

A PROPOS DE LA PLATFORME

Qu'est ce qui nous rend différent

Le diagram suivant représente le système de d-voting du point de vue du déploiement. Il décrit les composants et leurs interactions.

Ce site web héberge l'interface d'un système de vote. Ce système exécute des contrats intelligents, gérés par un ensemble de nœuds byzantins tolérants aux pannes.

Lorsqu'un administrateur crée un sondage, les paramètres du sondage sont sauvegardés sur une blockchain ainsi que toutes les transactions suivantes (fermeture/annulation du sondage, vote, etc.).

Une clé distribuée est générée au moment de la création de l'élection de sorte que lorsqu'un utilisateur vote, son vote est crypté avec la clé garantissant l'anonymat du vote. Cependant, le système ne garantit pas l'anonymat de l'électeur.

Lorsqu'un sondage est clôturée, les nœuds mélangent les bulletins de vote et vérifient leur exactitude avant de décrypter le mélange et de publier le résultat du sondage sur un contrat intelligent.

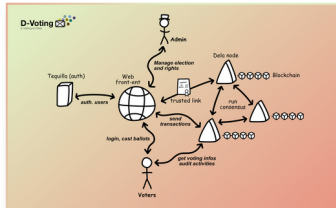


Figure B.2: The about page in French.

ÜBER DIE PLATTFORM

Was uns auszeichnet

Das folgende Diagramm zeigt das d-voting-System aus der Perspektive der Bereitstellung. Es beschreibt die Komponenten und ihr Zusammenspiel.

Diese Website beherbergt die Schnittstelle zu einem Evoting-System. Dieses System führt intelligente Verträge aus, die von einer Reihe von byzantinischen, fehlertoleranten Knoten verarbeitet werden.

Wenn ein Administrator ein Formular erstellt, werden die Parameter des Formulars auf einer Blockchain gespeichert, ebenso wie jede folgende Transaktion (Schließen/Abbrechen des Formulars, Abgabe einer Stimme usw.).

Bei der Erstellung des Formulars wird ein verteilter Schlüssel erzeugt, so dass bei der Stimmabgabe eines Benutzers seine Stimme mit dem Schlüssel verschlüsselt wird, der die Anonymität der Stimme garantiert. Allerdings erzwingt das System nicht die Anonymität des Wählers.

Wenn ein Formular abgeschlossen ist, mischen die Knoten die Stimmzettel und prüfen ihre Korrektheit, bevor sie die Mischung

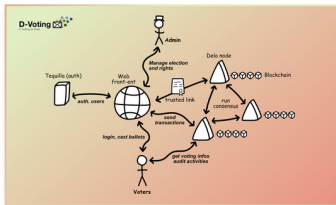


Figure B.3: The about page in German.