École Polytechnique Fédérale de Lausanne

D-voting - front-end development for an e-voting platform

by Capucine Berger-Sigrist
Badr Larhdir

Master Semester Project Report
DEDIS LAB

Prof. Bryan Ford
Responsible

Noémien Kocher
Project Supervisor

June 10, 2022

Contents

Chapter 1

Introduction

To answer to the rising place that e-voting has taken in our lives, the DEDIS lab has been working alongside students to develop new solutions. Based on the Dela blockchain, the D-Voting [2] project is an open platform that aims at providing a system guarantying anonymity of the votes, while being fully auditable and decentralized. As for the communication with the blockchain itself, a web application was developed to facilitate these interactions. Thus, as the third iteration of this project, the goal is to improve on the pre-existing functionalities and to implement new features in the front-end.

Chapter 2

Background

To give a general idea of how the D-Voting platform work we will first introduce its structure. The system is made of two back-ends, one that implements all the blockchain logic, and another, which we will refer to as the "web back-end", that authenticate and sign all the communications between the front-end and the proxies. The final part is a front-end to interact with the whole system, that is to say, authenticating users, managing elections and rights, voting and more generally sending transactions to the back-end.

In the beginning of the semester, the web API was not compatible with the back-end API, thus both part could not communicate since it was not possible to send the correct transactions to the back-end. Apart from the login with Tequila, most functionalities were outdated, but we were able to re-use parts of the code. In terms of User Interface (UI), not all visual elements were coherent across the pages as different libraries were used to represent the same components.

Chapter 3

Implementation

In this section, we are going to see in more details what we implemented during this semester. Our focus was at first turned toward having the main functionalities working but having a UI as close to "production ready" as possible was also an important part of the project.

3.1 Code structure

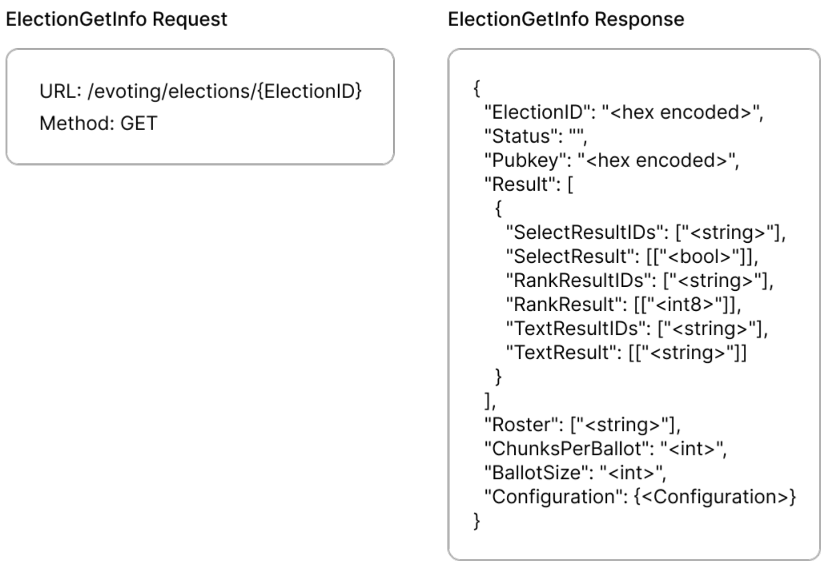
In a React app there are no specific structures on where to put files, nevertheless some common approaches in the ecosystem exists and we tried to follow some of them to better serve our application. The code structure follows a *grouping by file type* where each type follows a *grouping by features*, making it easier to access a specific type page (e.g. Admin) to change or add a chosen feature (e.g. `AdminTable`).

For the frameworks and libraries, we switched from the Material UI framework [4] to a less complete but extremely customizable UI framework called TailwindCSS [9], making our components easier to customize if they do not exactly suit our User Experience (UX) expectations.

In addition to that, we added two more libraries to solve the majority of our UI and UX assumptions. The first one being Headless UI [7] which are completely unstyled and fully accessible UI components designed to integrate seamlessly with Tailwind CSS (e.g Dropdown, Modals, Popover etc.). And finally Heroicons library [8], which is a set of 450+ free MIT-licensed SVG icons.

3.2 API

3.2.1 Updating the API calls



ElectionGetInfo Request

```
URL: /evoting/elections/{ElectionID}
Method: GET
```

ElectionGetInfo Response

```
{
  "ElectionID": "<hex encoded>",
  "Status": "",
  "Pubkey": "<hex encoded>",
  "Result": [
    {
      "SelectResultIDs": ["<string>"],
      "SelectResult": [["<bool>"]],
      "RankResultIDs": ["<string>"],
      "RankResult": [["<int8>"]],
      "TextResultIDs": ["<string>"],
      "TextResult": [["<string>"]]
    }
  ],
  "Roster": ["<string>"],
  "ChunksPerBallot": "<int>",
  "BallotSize": "<int>",
  "Configuration": {<Configuration>}
}
```

Figure 3.1: Example of an API call to get the information of an election and the response.

One of the biggest inconsistency in the API was the fact that it was not following the HTTP Methods conventions for a REST architecture. Indeed, any call made to the API was under the GET request type, which meant that it had to **retrieve resource representation/information only**, which was not always the case (e.g creating an election or casting a ballot). Hence, with the help of the back-end team and Noémien, we updated the whole API documentation. First, by following the HTTP Methods conventions and then cleaning up all the endpoints naming and input/output transactions, most notably in the election workflow involving the three actors (Smart contract, DKG and Neff shuffle service). Some of the most important changes are listed below:

- When getting the information of an election, it used to return a JSON with a `Format` field which was outdated and has been replaced with the `Configuration` field. It now also returns information about the `BallotSize` and the `ChunksPerBallot`, as shown in Figure 3.1. We also made a distinction between getting the information of a single election and all elections, respectively with `ElectionGetInfo` and `ElectionGetAllInfo`). When retrieving all the elections the API now returns a list of a lighter version of the election information for scalability purposes.
- There were some remaining `Token` field in the signed request calls which we removed as we are using a session based authentication rather than token based authentication.
- Adding a JSON format in case of an error on the API request.

3.2.2 Mocking the API

In parallel to implementing and updating all the API calls needed for a perfect communication with the server, we mocked the back-end responses using the Mock Service Worker library [3]. This presents two main advantages, the first one being that it is possible to work with the front-end application without having an available back-end server (if the API changes in the meantime), but this will also facilitate future feature implementations.

3.3 Election

3.3.1 Creating an election

Before explaining the structure of the create election page, we will list all available features:

- Creating an election from scratch which can consist in nested subjects (with at most one sub-subject) that contains questions of type rank, select or text.
- Importing an election from a JSON file and modify it to one's needs.
- Previewing the ballot generated from the election configuration.

An election configuration is composed of an array of nested `Subject` elements that can contain various types of question elements such as a `Text`, `Rank` or `Select`. The `Configuration` type and all other interfaces used to compose an election configuration can be found inside the `configuration.ts` file under the `types` folder.

As explained in the features list, there are two ways of creating an election. The first is to create it manually from the interactive page by clicking on the available buttons to structure an election. The second way is to first import it from a JSON file and eventually modify it after. When importing an election configuration from a JSON file, the application has to check whether that schema is valid or not. There are two layers of validation in the `Election` component to check if the JSON schema is a valid election configuration (Figure 3.2). The first layer is the JSON Schema Validation done by the cross-platform *AJV* library [1] that checks, thanks to a JSON schema file (i.e `schema/election_conf.json`), that the data has no unwanted fields and that the structure of each element respects the schema defined in the `configuration.ts` file. The second layer is checking the consistency of the schema (i.e cross field validation), which is not possible with a JSON schema validator like AJV. To solve this problem we used the *YUP* library [5]. By calling the `configurationSchema` function in the `types/configurationValidation.ts` file, *YUP* checks whether the fields of each elements (question or subject) are valid (e.g the max is always greater than the min, the length of the field `Order` array is always equal to the sum of the number of elements in each subject etc.). If any of the two layers outputs an error, a modal appears to display the message for the user.

For converting a JSON file to a `Configuration` type or vice versa (i.e when exporting an election), two functions are available in the `schema/JSONparser.ts` file, `marshallConfig` and `unmarshallConfig`. The `unmarshallConfig` function is used to transform an object (i.e a JSON object) into a `Configuration` type object. Additionally, `marshallConfig` function is used to transform a `Configuration` object into a JSON object.

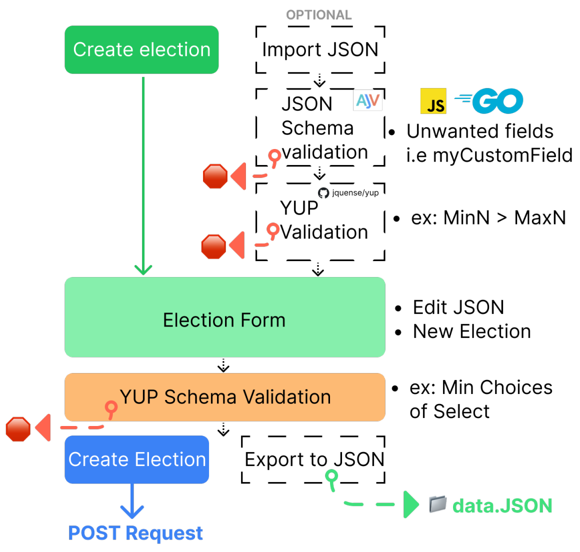
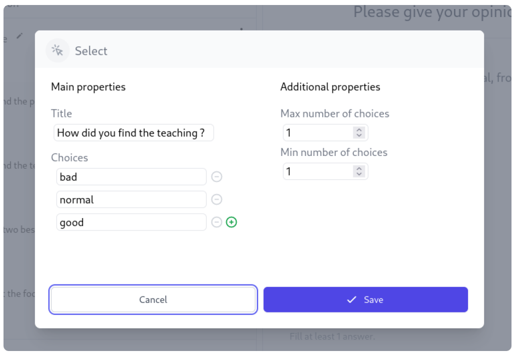


Figure 3.2: Schema for creating an election.

From the feedback we got during the mid-semester presentation we choose to redesign the election page (v2 during the semester) to simplify the UI and the UX. Indeed, we now have two separate views that are complementary to one another. As illustrated in Figure 4.1, the left part of the screen contains all the necessary tools to create an election and the right part contains a preview of an interactive ballot generated from the configuration of the left part. When adding a question to a specific subject, the modal `AddQuestionModal.tsx` appears and contains all the elements needed to complete this specific question type (see Figure 3.3).


Select

Main properties

Title: How did you find the teaching ?

Choices: bad, normal, good

Additional properties

Max number of choices: 1

Min number of choices: 1

Figure 3.3: Modal for adding a Select Question to an election Subject.

As we saw from the architecture of a configuration and from the Figure 4.1, an election can have a very deeply nested structure (e.g a question in subject of a subject of a subject). To maintain a visually pleasing and easy to use application, we opted to only have one level of nesting in the election configuration (i.e a sub-subjectt). Moreover since we can have, in theory, a very nested object as a configuration, the solution for updating the configuration state of the deeply nested components was to have a callback function `notifyParent` passed to the children components (`Subject`, `Text`, `Select` or `Rank`) to notify the parent component of any change in the state of the current child. This functions therefore updates the parent state from the child component.

Finally when the create election button is triggered and outputs no error, the user is redirected to the election page after closing the notification modal that notifies the user of the successful creation of the election.

3.3.2 Managing an election

There used to be two ways to manage the elections, the first one was via a table displaying all the existing elections and the second was in a dedicated page for each election. Basically, both pages offered the same views, with the election title, its status and the associated actions. We decided to simplify the former, replacing it with the `ElectionTable` component, by keeping only a subset of these actions, more specifically voting and seeing the results. Thus, the purpose of the table really is to offer an overall view of all the elections to the user, and eventually have access to the full details by clicking on the title of an election. As for the election page in `election/Show.tsx`, it now also displays the status of the nodes associated to the election and a status timeline, showing the current status of the election along with the previous and next states. Both can be found, respectively, in the components `DKGStatus` and `StatusTimeline`. In the following subsections we will see more precisely what needed to be implemented in the hook `useChangeAction` to have a fully working election flow and how to transition from one status to the next, as shown in Figure 3.4.

Node status

We realized a tad late in the semester, that a major functionality was still missing for the election flow to work in its entirety. Indeed, to open an election, a user must first initialize and setup the roster of nodes of the election. The initialization required to send a POST request to each of the proxy associated to the nodes. The DKG service then created the actors. Since the service does not notify us when the action finished, we had to implement a polling mechanism (which can be found in `PollStatus.ts`) in the front-end. The polling consists in sending GET requests, continuously, and directly to the aforementioned proxy (rather than the default proxy address) until the status of a node changed. If the user leaves the page during this phase, the

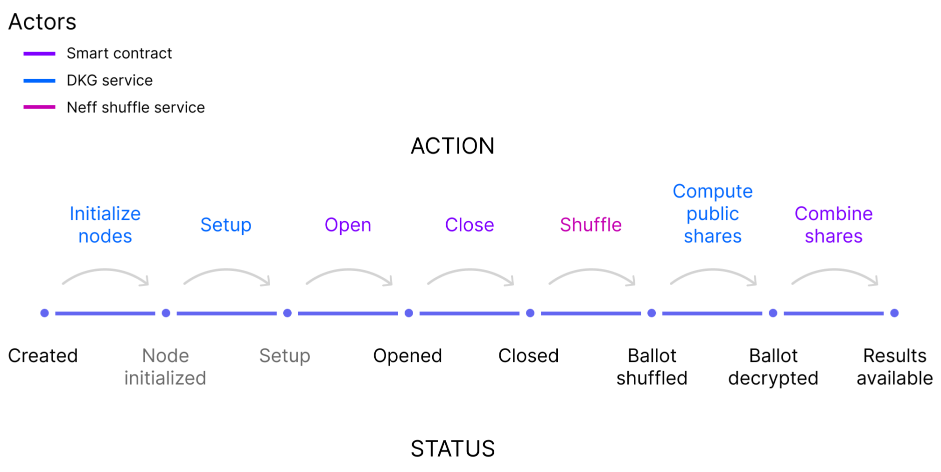


Figure 3.4: The election flow. The figure shows each of the election status and the actions, with their actors, necessary to transition from one to the other. The status *Node initialized* and *Setup* are internal statuses, computed by polling the DKG service.

`AbortController` sends an `AbortSignal` to stop the polling. This solution might not be optimal and while it would need some rework, a cleaner way might be to do the polling in the web back-end, so that the results could be cached and served to multiple clients. We also use the local storage of the browser to record if an action is on-going, if so, when the user comes back on the page, the polling restart automatically.

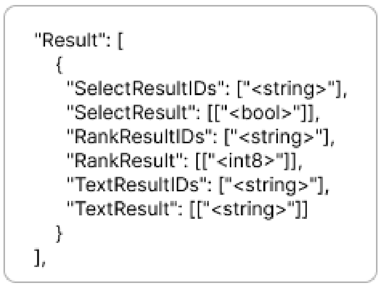
As for the setup, the user is invited to select one of the proxy associated to the roster using the modal `ChooseProxyModal`. This choice is also stored temporarily in the local storage (in case the user were to leave the page while the setup was on-going). A PUT request is then sent to the DKG service, which sets up all the nodes of the roster. Similarly to the initialization, we poll the status of the node associated to this proxy until the action is finished. Once this is done, a user can then open the election.

Election status

While the actions after the setup were already implemented, we had to adapt them all to be compatible with the updated API calls. But having the skeleton of the code certainly helped. Like for the nodes, to make sure the action was correctly executed, we poll the status of the election by getting its information from the back-end, and stop when it matches the expected status. Finally, we were able to have a meeting with Lindo Duratti, who is in charge of the actual e-voting system at EPFL, and he gave us some very insightful feedback on the current functionalities of the front-end. One of the comment he made was about the ability to make a test run of an election before officially opening it. To answer to this need, we implemented a new feature, which is being able to delete an election completely at any point in time.

3.3.3 Result of an election

Once all the shares of the nodes have been combined, we obtain the result by making the API call `ElectionGetInfo`, which returns, among other things, the `Configuration` and the `Result` objects. The former contains all the structure of the election and can be parsed recursively to display the subjects with their questions and their choices. The latter is an array of answers for each of the ballot cast, as shown below.



```
"Result": [
  {
    "SelectResultIDs": ["<string>"],
    "SelectResult": [["<bool>"]],
    "RankResultIDs": ["<string>"],
    "RankResult": [["<int8>"]],
    "TextResultIDs": ["<string>"],
    "TextResult": [["<string>"]]
  }
],
```

Figure 3.5: Schema type for the `Result` object.

This schema was admittedly not very practical for the aggregation of the results, especially since `JavaScript` does not natively implement the functions `groupBy` or `aggregate`. For select and text question, the results were counted as:

$$\frac{number\_of\_votes\_for\_a\_choice}{total\_number\_of\_ballots\_cast}$$

When a user ranks their choices from 1 to N (lower is better) , we translate this into them attributing to each, points from 0 to N-1, thus we obtain the following formula for the rank questions:

$$1 - \frac{sum\_of\_points\_of\_a\_choice}{total\_number\_of\_points\_attributed}$$

Finally, for transparency reasons, the way ballots are counted is disclosed on the `Result` page, which is accessible by everyone.

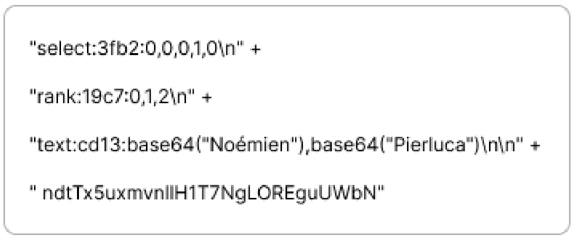
3.4 Casting a ballot

This feature required some major modifications due to the API changes. Getting the election information, we took inspiration from `Golang`, as we receive the `Configuration` in JSON format and `unmarshal` it into an object of type `Configuration`. In parallel, while we parse the JSON, we also create an `Answers` object which contains a mapping between each question's ID and an array to hold the future answers of the voter. We tried to decouple as much as possible the part that handled the rendering from the part that handled the logic of casting a vote, this was useful most notably to create the preview in the creation of an election. Thus, we ended up with the two main components `Ballot` and `BallotDisplay`.

Once a user is logged in, they can vote on the election, as shown in Figure 4.3. For each type of question, there is a component, either `Rank`, `Select` or `Text`, which handle the rendering and the logic to verify the inputs of the user. Previously, the user was not redirected to another page once their vote was cast. Thus we tried two different way, the first one was to redirect them home immediately after they clicked on the cast vote button and to display a `Flash message`, but some early feedback indicated that they weren't sure if their ballot had been taken into account. Instead we now have a modal, saying that the vote was successfully cast and redirects the user to the election table or the election page, depending on where he was before voting, on close.

3.4.1 Ballot encoding and encryption

As mentioned in the earlier reports, the `kyber` library we use can only encrypt up to 29 bytes at a time. So to solve this problem we first encode each vote, using the function `voteEncode`, by concatenating the answers, as shown in Figure 3.6. To maintain anonymity, all the ballots for an election must have the same length and must be a multiple of 29. This information is computed in the back-end when an election is created, and made available by calling `ElectionGetInfo`, which gives us the `BallotSize` and the `ChunksPerBallot`. Thus, we add a padding of random characters to have an encoded ballot of size `BallotSize` and divide it into `ChunksPerBallot` number of chunks. Each chunk then gets encrypted using the ElGamal encryption system, producing an array of ciphertexts [K, C] that can be sent to the back-end.



```
"select:3fb2:0,0,0,1,0\n" +

"rank:19c7:0,1,2\n" +

"text:cd13:base64("Noémien"),base64("Pierluca")\n\n" +

" ndtTx5uxmvnllH1T7NgLOREguUWbN"
```

Figure 3.6: Encoding of a ballot containing the answer of a select, a rank and a text question. The last line shows the padding necessary, so that the ballot can be divided uniformly into chunks of 29 bytes.

3.5 Admin

As its name indicates, the Admin page is only accessible to the administrators, and enables them to manage the more technical aspects of the system.

3.5.1 Managing user roles

To allow some users to create or audit elections we need a user role managing section. This section is available as a table in the Admin page (Figure 4.5), which makes it possible to add or remove users' role to the admin database. There are two administrative roles, the admin and the operator. The difference between them is that an operator cannot manage roles of users while an admin can add or remove user roles. Either way, both of the roles can create or audit elections.

3.5.2 Managing the proxy addresses

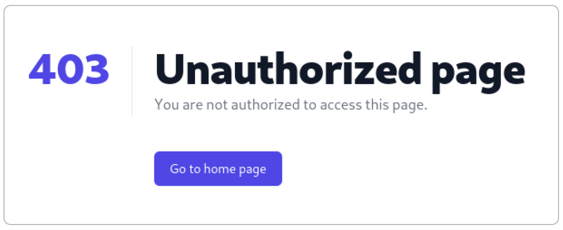
Some of the actions during an election require the requests to be sent directly to the proxy address associated to a particular node. Since asking the user for this address each time would be quite cumbersome, we implemented in the web back-end a database which stores the mapping between each node and its proxy. Thus the entry in this database can be managed from the front-end using a table. The admin can then create a mapping, edit a pre-existing one or completely delete an entry.

3.6 Other features

In this section we will discuss some smaller but yet important features implemented during the project. For managing information of current users (logged, name, role), the application implements a global state containing the authentication state called context (named `AuthContext`). The `AuthContext` is a way to manage state globally, without having to pass props to components. This is especially useful for authorizing sensitive content to a specific set of users (e.g Create Election page). We fixed some vulnerabilities that made it possible for an unauthenticated user to access not only authenticated routes but also role sensitive routes such as the Admin page or the Create Election page.

Moreover some pages where not protected by roles, so we made sure that each page available in the application has the correct authorized roles to access it. On the same fix we added an access denied page (see Figure 3.7), if a connected user wants to access a route that is not originally accessible to them (e.g Admin page for voter user).

To correctly notify a user of any informative information, we implemented two UI components (modals and flash messages) that makes the user more aware of any changes that could have been made in either the front-end of the application (e.g creating an election) or in the back-end (e.g fetching the elections). The flash message has a specificity of being able to display different type of notifications with different background colors for a better user experience. Therefore, information flash messages (Figure 3.8) are displayed in a blue background on the

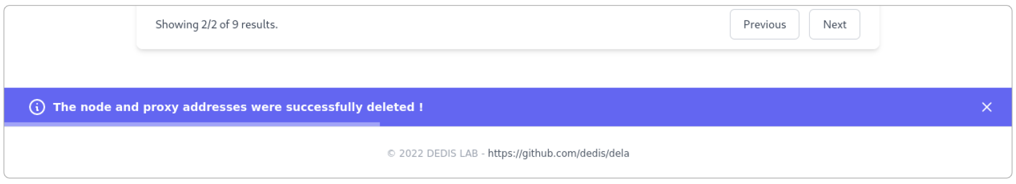

403

Unauthorized page

You are not authorized to access this page.

Figure 3.7: Unauthorized page. This page is displayed when a user tries to access a page for which they do not have the necessary authorization (triggering an error 403).

bottom of the application, whereas warnings or errors are, respectively, displayed in an orange or red background to better emphasize the importance of the message to the user.

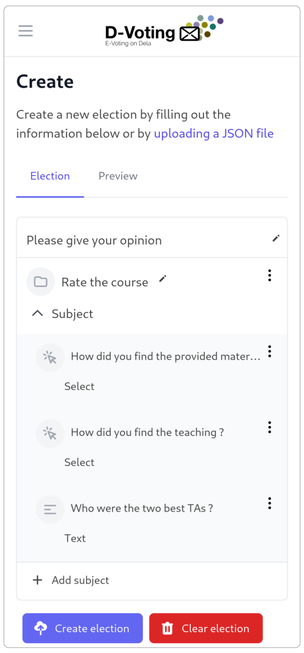

Showing 2/2 of 9 results.

The node and proxy addresses were successfully deleted !

© 2022 DEDIS LAB · https://github.com/dedis/dela

Figure 3.8: Example of an information flash message.

Finally, our app is a responsive web application, which allows the users to have the same user experience even when using different devices. Indeed, as the device changes, the format of the content displayed changes accordingly since we made sure that any UI component could be correctly displayed in any device format.


D-Voting
E-Voting on Dela

Create

Create a new election by filling out the information below or by uploading a JSON file

Election Preview

Please give your opinion

Rate the course

Subject

How did you find the provided mater…

Select

How did you find the teaching ?

Select

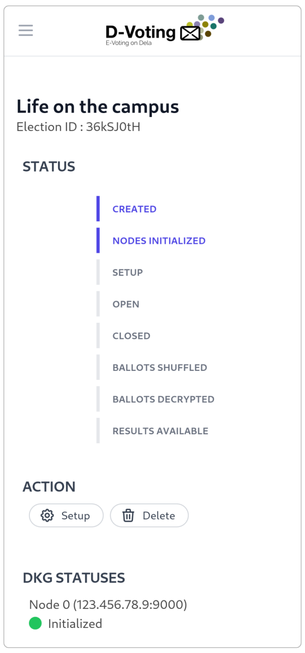
Who were the two best TAs ?

Text

Add subject

Create election Clear election

(a) Create election.


D-Voting
E-Voting on Dela

Life on the campus

Election ID : 36kSJ0tH

STATUS

CREATED
NODES INITIALIZED
SETUP
OPEN
CLOSED
BALLOTS SHUFFLED
BALLOTS DECRYPTED
RESULTS AVAILABLE

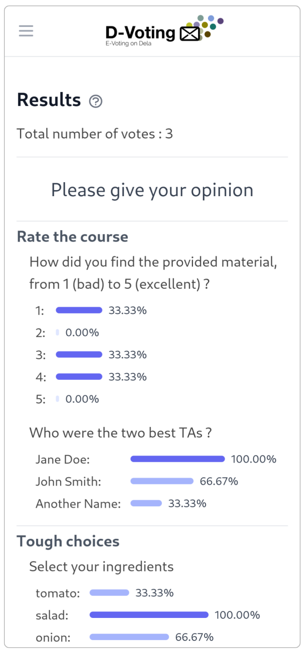
ACTION

Setup Delete

DKG STATUSES

Node 0 (123.456.78.9:9000)
● Initialized

(b) Election page.


D-Voting
E-Voting on Dela

Results

Total number of votes : 3

Please give your opinion

Rate the course

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

1:	33.33%
2:	0.00%
3:	33.33%
4:	33.33%
5:	0.00%

Who were the two best TAs ?

Jane Doe:	100.00%
John Smith:	66.67%
Another Name:	33.33%

Tough choices

Select your ingredients

tomato:	33.33%
salad:	100.00%
onion:	66.67%

(c) Results page.

Figure 3.9: Examples of views on a mobile device.

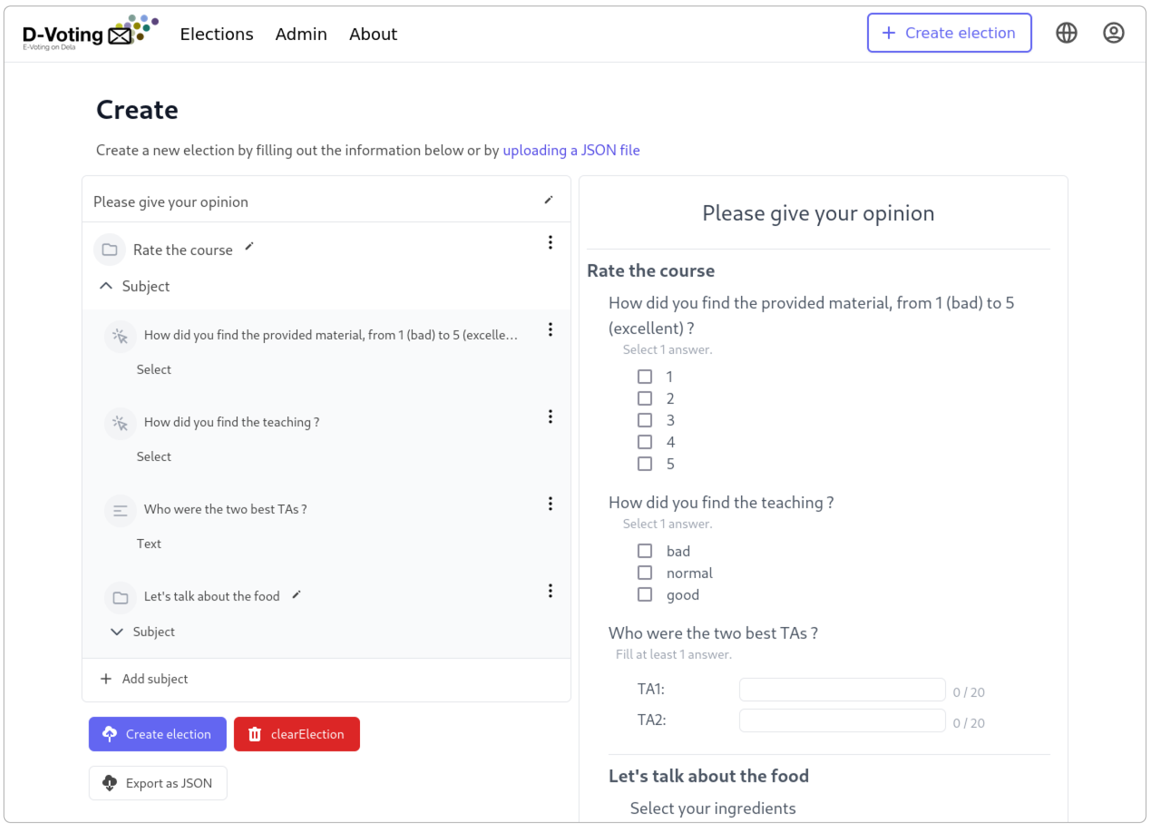
Chapter 4

Results

Now that we have seen quite into detail how we implemented each feature during the semester, we believed that the most appropriate way to show our results was with a series of pictures, rather than a long-winded paragraph. After all a picture is worth a thousand words! As for the usability aspect of the front-end, we left our fate in the hand of the users who interacted with it during the UX testing phase.

4.1 Current state of the front-end

Figure 4.2 to 4.5 show different views of the system as it currently is.


D-Voting
E-Voting on Dela
Elections Admin About
+ Create election

Create

Create a new election by filling out the information below or by uploading a JSON file

Please give your opinion

Rate the course

Subject

How did you find the provided material, from 1 (bad) to 5 (excelle…

Select

How did you find the teaching ?

Select

Who were the two best TAs ?

Text

Let's talk about the food

Subject

+ Add subject

Create election clearElection

Export as JSON

Please give your opinion

Rate the course

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

Select 1 answer.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

How did you find the teaching ?

Select 1 answer.

- ☐ bad
- ☐ normal
- ☐ good

Who were the two best TAs ?

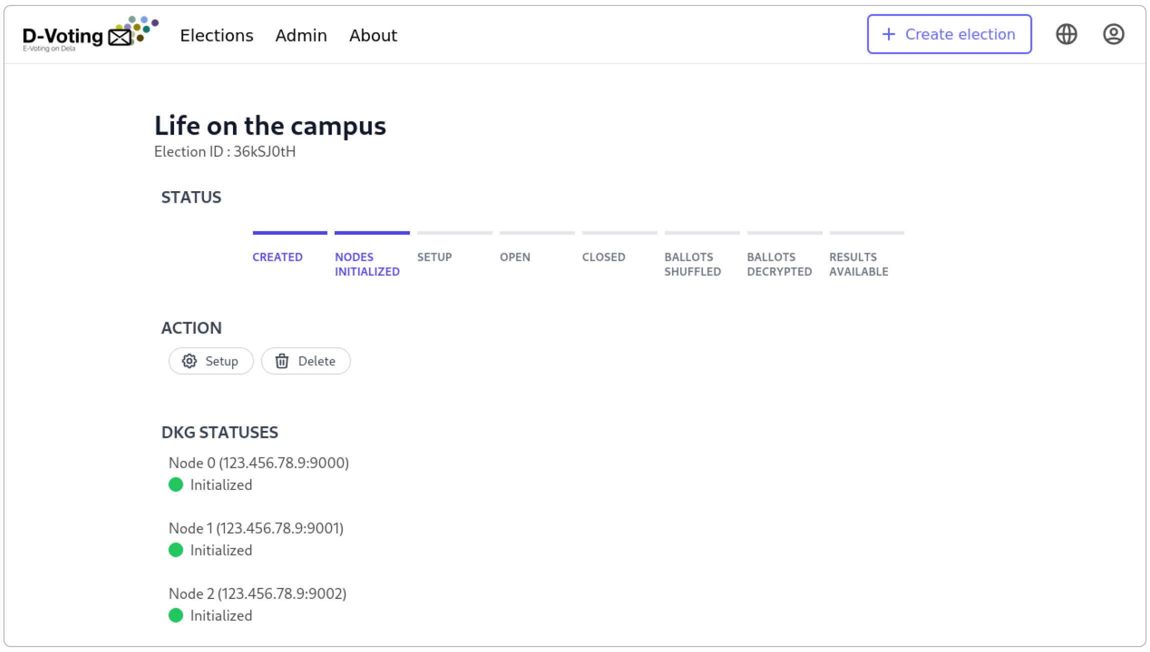
Fill at least 1 answer.

TA1:		0 / 20
TA2:		0 / 20

Let's talk about the food

Select your ingredients

Figure 4.1: View of the creation of an election. The left part contains the actual elements to create an election and to input question, the right part shows the preview.


D-Voting
E-Voting on Dela
Elections Admin About
+ Create election

Life on the campus

Election ID : 36kSJ0tH

STATUS

CREATED NODES INITIALIZED SETUP OPEN CLOSED BALLOTS SHUFFLED BALLOTS DECRYPTED RESULTS AVAILABLE

ACTION

Setup Delete

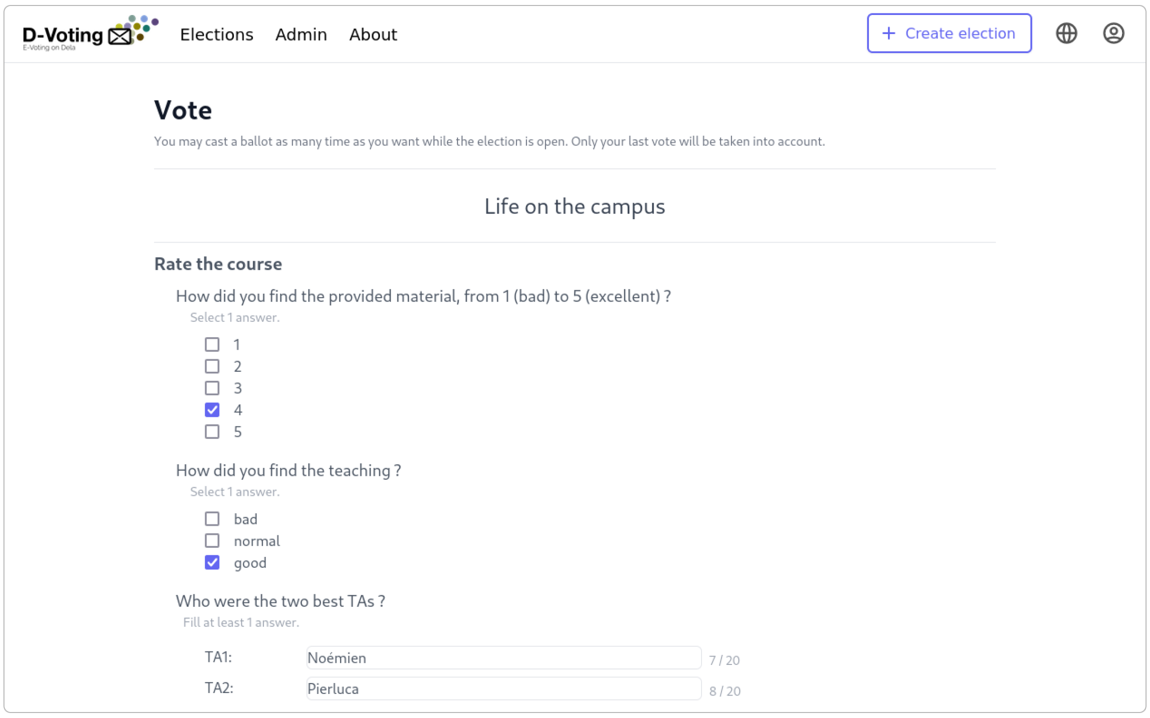
DKG STATUSES

Node 0 (123.456.78.9:9000)
● Initialized

Node 1 (123.456.78.9:9001)
● Initialized

Node 2 (123.456.78.9:9002)
● Initialized

Figure 4.2: View of the Election page. The current status of the election is displayed along with the available actions. The address of each node of the roster and its status are shown underneath.



Vote

You may cast a ballot as many time as you want while the election is open. Only your last vote will be taken into account.

Life on the campus

Rate the course

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

Select 1 answer.

- ☐ 1
- ☐ 2
- ☐ 3
- ☑ 4
- ☐ 5

How did you find the teaching ?

Select 1 answer.

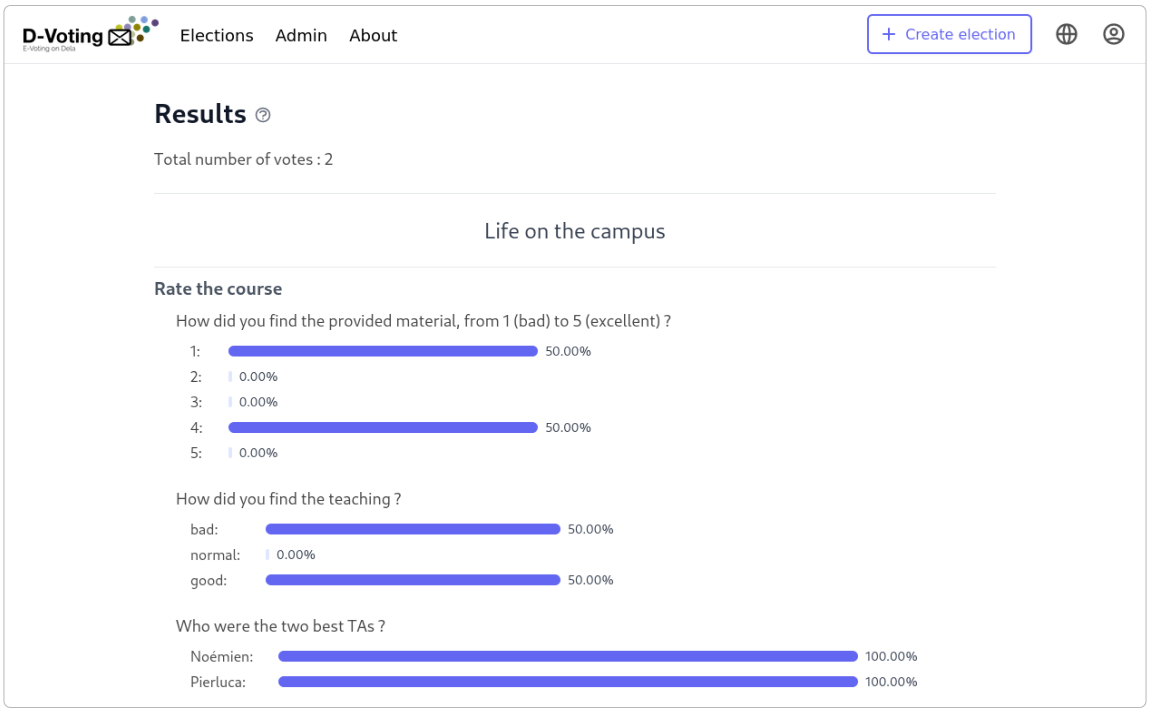
- ☐ bad
- ☐ normal
- ☑ good

Who were the two best TAs ?

Fill at least 1 answer.

TA1:	Noémien	7 / 20
TA2:	Pierluca	8 / 20

Figure 4.3: View of casting a ballot. It shows a subject containing two select and a text questions.



Results

Total number of votes : 2

Life on the campus

Rate the course

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

1:	50.00%
2:	0.00%
3:	0.00%
4:	50.00%
5:	0.00%

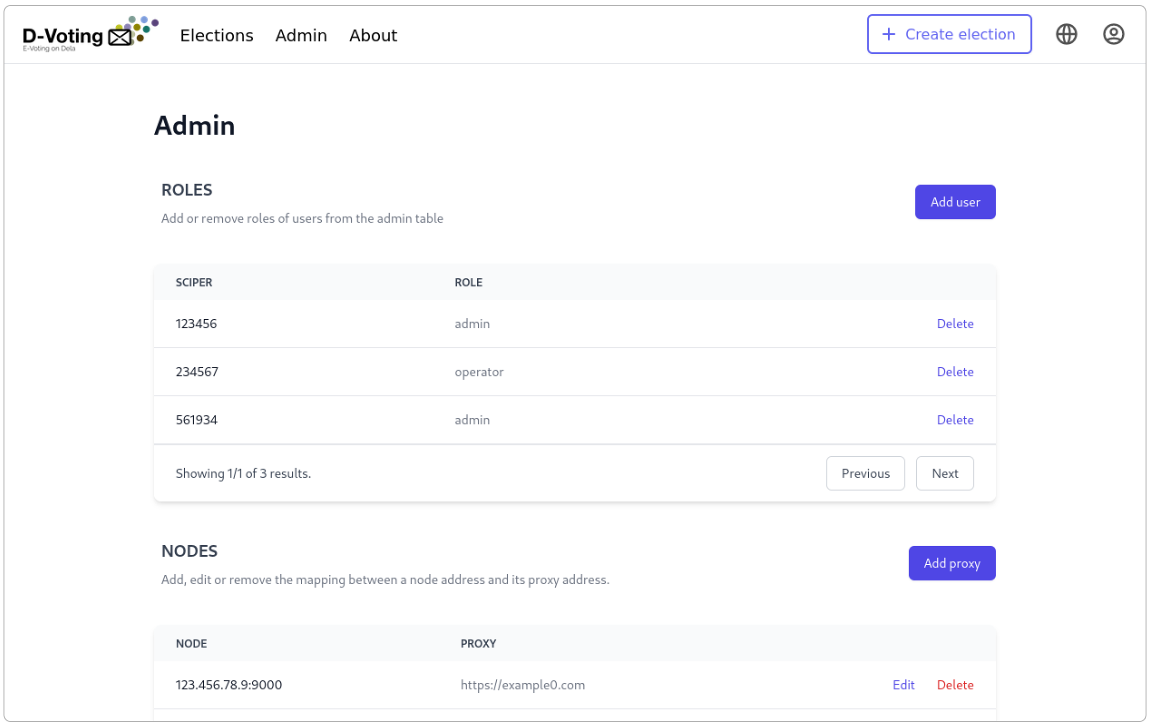
How did you find the teaching ?

bad:	50.00%
normal:	0.00%
good:	50.00%

Who were the two best TAs ?

Noémien:	100.00%
Pierluca:	100.00%

Figure 4.4: View of the Result page. Explanation about the results can be found in the Popover by clicking on the button "(?)" next to the title of the page.



Admin

ROLES

Add or remove roles of users from the admin table

Add user

SCIPER	ROLE	
123456	admin	Delete
234567	operator	Delete
561934	admin	Delete

Showing 1/1 of 3 results. Previous Next

NODES

Add, edit or remove the mapping between a node address and its proxy address.

Add proxy

NODE	PROXY		
123.456.78.9:9000	https://example0.com	Edit	Delete

Figure 4.5: View of the Admin page. It displays two tables, one to manage the user, another to manage the nodes and their proxy addresses.

4.2 User experience testing

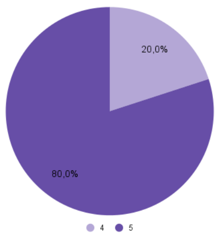
We were made to understand very early in the project that this would be an important part. And after conducting a small UX testing campaign, we could not agree more! Seeing how other users interact with the system was both fun to observe and very insightful. As, we finished implementing the last features a bit late, we were greatly constrained by time, so we were not able to have as many people as we would have liked, and the sample of user is made entirely of students. But we still managed to have a total of 13 people. Running the tests with the fully mocked API, we asked 4 users to act as admins and to accomplish the following tasks:

1. Login.
2. Go to the admin page and add themselves as Admin and delete another user role.
3. Modify the proxy address of the node 213.456.78.9:9006 (or any node not on the first page of the table).
4. Create an election containing at least one subject with each type of question (rank, select and text).

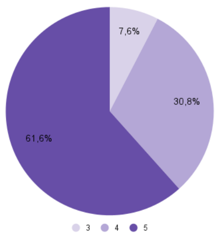
5. Open the election "Life on the campus" and cast a vote on this election.
6. Make the result for "Life on the campus" available and go see the results.
7. Complete a Google form to give their feedback (We had this brilliant idea, if we do say so ourselves, to actually get the feedback by making the users answer directly on the D-Voting platform, unfortunately we were not able to get a server up and running on time).

And 9 other users tested the system as operators, they accomplished the same tasks except for number 2 and 3, which they skipped.

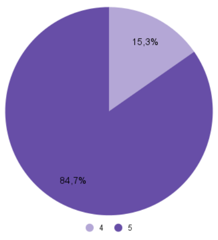
In the end, we asked them, for each page, the two following questions: whether they liked the UI, how easy was it to use, and to justify if they would have changed something. Overall, the users tended to have a very positive experience with the system, as shown in Figure 4.6, and some of the comments were very thoughtful. Although, due to the very subjective nature of the first question, we had without any surprise, both extreme of the spectrum. Some liked the simplicity of the design, and that the few accents of colours really helped them finding the important actions, while other found the UI a bit too sparse and empty, and would have enjoyed a more colourful website.



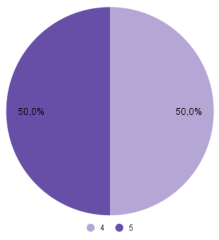
(a) Admin page.



(b) Election create page.



(c) Vote page.



(d) Election page.

Figure 4.6: Results of the question *"On a scale from 1 to 5 (1=not at all, 5=very much), was the interface easy to use ?"*, for each of the pages. All users gave at least a 3 to all pages, and most of them rated them a 5.

In the end, these are the main takeaways of the tests:

- For the Admin page, a user pointed out that it would be great to have the possibility to choose the number of items displayed on a single page of the tables (e.g. 5, 10 or "See all"), as it would make searching for a Sciper or a node easier.
- For the creation of an election, people struggled a bit sometimes to understand its structure (most notably the role of the subject) and how to create it. The users felt that it would be great to have more explanation, tooltips or a tutorial to help them. Some smaller comments also involved moving the `Min number of choices` before the max in the modals to create a question. And a lot of users where perturbed by the remaining `Object ID` that appeared in the error messages and in the modal when the election was successfully created.

- For the Vote page, one person found the ranking to be less intuitive than the other types of questions. Another user also miss-clicked and went back on the Election page rather than casting their vote. It might be a good idea to have a modal asking the user if they are sure they want to leave the page.
- As for the Election page, some users also struggled trying to change the status of the election (and tried to click directly on the status timeline). They also felt that for an operator type of user, it would be better to have a simpler election flow (i.e. if a single action did more). For example clicking on the action Open would initialize the node, set them up and open the election at the same time. Or at the very least they would like more information on the status.

Chapter 5

Improvements and future work

One major improvement that could be made is in the way errors are handled. For the moment, the error messages are admittedly not very user friendly. They suffer from being not helpful enough (the message is very generic) and too precise (the entire error content is displayed) at the same time. The latter is actually quite useful for debugging but makes the message unreadable for a normal user. One possible solution, at least in the short term, would be to have a "More information" drop-down or link which would hide by default the full error code from the user but would still give the possibility to read it. Also since the flash messages came later in the project, some error trigger a flash message while other still display a modal. Uniforming this would greatly improve the coherence of the front-end. Another question that arose was what to do with the home page, as it is currently quite empty. We discussed making a "Shepherd tutorial" [6] but the system currently has quite literally four buttons on which the user can click on, making this option a bit overkill.

Another small modification would be to change the name of the page and navigation tab for the elections, as we aren't running election only anymore. We thought that "surveys" or "polls" would be more appropriate. Finally, as detailed earlier in Section 4.2, we should take into account the results of the UX testing to improve the existing functionalities.

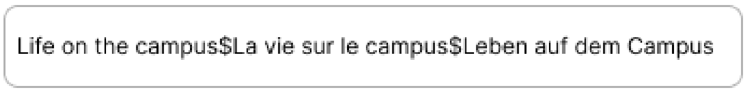

Life on the campus$La vie sur le campus$Leben auf dem Campus

Figure 5.1: Example of the `MainTitle` field of a configuration after concatenating the three inputs for English, French and German using the "$" as a character delimiter.

As for new features, there are three major points that the current e-voting platform has that are still missing in our system. The first one is to have French and German translations of the interface but also in the creation of an election. The former should only require to take all the keys from the file `en.json` and translate the entries, using Google translate for example, and pasting the results in the `fr.json` and `de.json` files. For the latter, we discussed about a way to avoid duplicating fields in the back-end, and the solution would be to implement the translation completely in the front-end. Basically, during the creation of an election, three input fields would be displayed (one for each language) for each title and choices of a question. The text inputs of these three fields would then be joined, using a character delimiters, in the `Configuration` object and sent to the back-end, such an example is shown in Figure 5.1. This would then require to change slightly the way the configuration is used to display the content of the election. The second missing feature is a more fine-grained control for user roles, this idea was already mentioned in the previous report but this would enable an admin to open an election only to a subset of the users (a section or faculty for example). The last point is that for the moment, a user cannot easily verify if their vote has been taken into account. To remedy this, we could generate a "ticket" that the user could download once they voted. They would then be able to upload it to assert if the vote was correctly cast.

Finally here are additional nice-to-haves to simplify the User Interface for the users:

- The possibility of redirecting to the current page after logging in with Tequila whenever a user tries to enter an authenticated page while not being logged in (e.g casting a vote from the URL).
- During the creation of an election, when adding a subject, it would be nice to have a modal to add directly all the question elements needed for the specific subject, rather than clicking each time on the top right corner of the subject to add another question.
- Having more flexibility when ranking answers in a rank question when casting a ballot. Instead of having to rank every choices, the user could then rank n items among the total numbers of choices.
- The possibility to download the results of an election in CSV rather than JSON, although it requires to restructure the schema of the result to have a simpler visualization of the CSV file.
- Having more fields in the `Configuration` of an election, such as a hint or an expected answer for an input in a text question, would make the creation of an election easier. Asking the user for a Regex is not very friendly.

Chapter 6

Conclusion

The purpose of this project was to improve on the front-end of the D-Voting platform. As some of the main functionalities necessary to run voting instances on the blockchain were missing at the start of the semester, we first focused our attention of this part of the project. And we are proud to announce that a full election can now be run from the front-end, from its creation down to seeing its result and naturally casting a vote. In parallel, we also did an important redesign of the UI, which according to the feedback from the UX testing seems to work quite well.

Bibliography

[1] *Ajv JSON schema validator*. original-date: 2015-05-19T23:23:32Z. June 10, 2022. URL: https://github.com/ajv-validator/ajv (visited on 06/10/2022).

[2] *D-Voting*. original-date: 2021-06-10T11:26:01Z. May 6, 2022. URL: https://github.com/dedis/d-voting (visited on 06/10/2022).

[3] *Mock Service Worker*. original-date: 2018-11-13T14:58:44Z. June 10, 2022. URL: https://github.com/mswjs/msw (visited on 06/10/2022).

[4] *MUI Core*. original-date: 2014-08-18T19:11:54Z. June 10, 2022. URL: https://github.com/mui/material-ui (visited on 06/10/2022).

[5] Jason Quense. *Yup*. original-date: 2014-09-22T23:54:22Z. June 10, 2022. URL: https://github.com/jquense/yup (visited on 06/10/2022).

[6] *react-shepherd*. original-date: 2019-02-04T04:19:26Z. June 9, 2022. URL: https://github.com/shipshapecode/react-shepherd (visited on 06/10/2022).

[7] *tailwindlabs/headlessui*. original-date: 2020-09-16T09:53:09Z. June 10, 2022. URL: https://github.com/tailwindlabs/headlessui (visited on 06/10/2022).

[8] *tailwindlabs/heroicons*. original-date: 2020-02-24T14:16:00Z. June 10, 2022. URL: https://github.com/tailwindlabs/heroicons (visited on 06/10/2022).

[9] *tailwindlabs/tailwindcss*. original-date: 2017-10-06T14:59:14Z. June 10, 2022. URL: https://github.com/tailwindlabs/tailwindcss (visited on 06/10/2022).

Appendix A

Uploading an election in JSON

```
{
  "MainTitle": "Please give your opinion",
  "Scaffold": [
    {
      "ID": "0xa2ab",
      "Title": "Rate the course",
      "Order": ["0x3fb2", "0x19c7"],
      "Subjects": [],
      "Ranks": [
            {
              "Title": "Rank the cafeteria",
              "ID": "0x19c7",
              "MaxN": 3,
              "MinN": 3,
              "Choices": ["BC", "SV", "Parmentier"]
            }
          ],
      "Selects": [
        {
          "Title": "How did you find the provided material, from 1 (bad) to 5 (excellent) ?",
          "ID": "0x3fb2",
          "MaxN": 1,
          "MinN": 1,
          "Choices": ["1", "2", "3", "4", "5"]
        }
      ],
      "Texts": []
    }
  ]
}
```

Listing 1: Example of a valid JSON configuration that can be uploaded to create an election.

Appendix B

Downloading election results in JSON

```
{
  "Title": "Please give your opinion",
  "NumberOfVotes": 2,
  "Results": [
    {
      "Title": "Rate the course"
    },
    {
      "Title": "How did you find the provided material, from 1 (bad) to 5 (excellent) ?",
      "Results": [
        {
          "Candidate": "1",
          "Percentage": "50.00%"
        },
        ...

      ]
    },
    {
      "Title": "Rank the cafeteria",
      "Results": [
        {
          "Candidate": "BC",
          "Percentage": "100%"
        },
        ...

      ]
    }
  ]
}
```

Listing 2: Example of the results in JSON when downloading them (the "..." are for the illustration only).