Master Semester Project
DEDIS lab

E-Voting on Dela Frontend

Sarah Antille

June 11, 2021

Responsible
Prof. Bryan Ford
EPFL/DEDIS

Supervisors
Noémien Kocher and Gaurav Narula
EPFL/DEDIS

Contents

1 Introduction

1.1 Motivation

While more and more of our life happens online, one challenge that hasn't quite be mastered yet in Switzerland is e-voting. While some cantons allowed voters to cast their vote online for a brief period before 2019 as part of a trial phase, it was ended in June 2019 [1]. The concept of e-voting might sound simple but its implementation is not. Indeed the main properties that are needed is the anonymity of the vote, a secure storage for the votes such that they can not be deleted or tampered with, and distributed trust such that a single node shouldn't be able to prevent a user to cast a vote.
Those properties are not trivial to guarantee and that has made e-voting an on-going research topic. However one of the technologies that seems promising is the distributed ledger technology which is the one that is used in this project.

1.2 Context

The DEDIS lab has been working lately on its own implementation of a distributed ledger architecture called DELA [2]. It offers the primitives to run distributed protocols and smart contracts, handled by a set of Byzantine fault-tolerant node. Their previous ledger called cothority/onet [3] had an e-voting system that has actually been used in the past for EPFL elections [4]. At the start of this semester, DELA was missing some pieces (such as a shuffling service, smart contracts that would hold election data, ...) to run an e-voting system. My semester project implements the frontend of an e-voting system while Anas Ibrahim and Vincent Parodi's semester project implements the backend of an e-voting system. The requirements of the e-voting system as a whole are the following :

- votes are anonymous
- a quorum of nodes must witness the whole process
- votes are securely stored and can not be tampered/deleted
- anyone can verify the election's result
- a single node can not prevent a user to cast his/her vote

The requirement that needs to be implemented by the frontend is the anonymity of the vote. All the others fall on the backend team. It is worth noting that the system does not guarantee the anonymity of the person who votes.

1.3 Goal

The goal of this project is to implement the frontend of a new E-Voting system based in Dela. As stated previously, the backend part is implemented by Anas Ibrahim and Vincent Parodi. The common goal of both projects is to have a first proof-of-concept of a simple E-Voting system that runs on Dela.

[1]https://www.ch.ch/en/demokratie/voting-online/how-do-i-vote-online/
[2]https://github.com/dedis/dela
[3]https://github.com/dedis/cothority/tree/main/evoting
[4]https://actu.epfl.ch/news/epfl-uses-blockchain-technology-to-secure-e-voting/

The evoting system should be usable by two actors, an administrator who is responsible for managing an election and a voter who is simply an end-user who can cast a vote and get the result of an election. Figure 1 describes the different steps of the scenario that is fulfilled by my frontend. The user who creates an election becomes the administrator of the election and is the only one who is able to trigger the actions in orange.

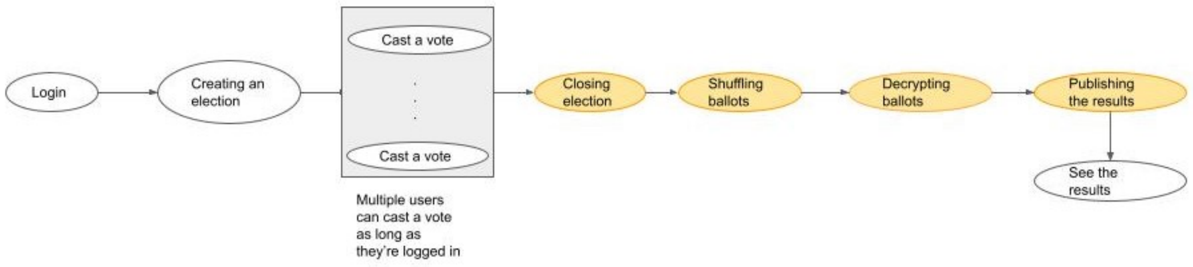


Figure 1: Basic scenario

A complementary goal is also to offer a nice user experience in terms of usability and overall impression.

2 Implementation

I decided to make a web-application simply because I was interested to learn more about web-development and widen my coding skills. Since I had no previous experience with it, I had no preference of which technology to use. After some research, I decided on using React (instead of Angular or Vue, two other popular frameworks) because of the community support it had and a learning curve easier than Angular for someone who is getting started with frontend. I used create-react-app [5] to create my web application to avoid doing the configuration myself. Since it created a single page React application but I wanted a multiple page application, I used React Router [6] as a routing solution.
Since React 16.8, hooks have been added [7], making it possible to write functional components instead of class and using hooks to deal with states which is the way I chose to go.

Instead of making two separate web application, one for an administrator and one for a voter, I decided on making only one with the reasoning that when authentication will be added, preventing a user to access some restricted webpages should be simple. Indeed to make that change in the future, the Status page should be made administrator-only and that's it.

For CSS styling I opted for basic CSS stylesheet, one for each react component that needed it.

2.1 Code structure

To be able to correctly fulfill the scenario from Figure 1, my web application provides 4 main pages:

1. a page to create an election
2. a page that would act as a sort of administrator dashboard to see all the elections that has been created, their status and the ability to trigger actions such as closing, canceling, shuffling, decrypting and getting the results

[5]https://create-react-app.dev/docs/getting-started
[6]https://reactrouter.com/
[7]https://reactjs.org/docs/hooks-overview.html

3. a page to cast a vote on on-going elections
4. a page to access the results of successful elections

There are also 2 other pages, a homepage and an about page to display some information about the election process. Because of the small number of actual web pages needed, I took the approach of structuring my code by grouping all the functional components that appeared on the same web page in a corresponding folder (namely `election-creation`, `election-status`,`voting`,`result-page`, `homepage`,and `about`. Other components such as a footer, the navigation bar, a login window that hides the content of a web page until the user logs in and a custom modal also have their own folder at the same level. All the custom hooks that I created along with some utility components which are usually used by at least two different web pages are grouped in the folder `utils`.
The hooks `useFetchCall`, `usePostCall`,and `useGetResults` all make API calls to the backend. They all relay one way or another the loading status and errors (if any) to the components from which they are used.

To correctly run, my code needs 3 running dela nodes whose addresses and public keys are hard-coded in the file `CollectiveAuthorityMembers.js` in the utils folder. As described in the README of the github repository of this project [8], this file might need to be updated every time the nodes are started. A threshold that indicates the number of nodes that must correctly complete the shuffle operation is also hard-coded in the same file and always need to be at least 2/3 of the running nodes (it is assumed at least 2/3 of the nodes running are not malicious).

2.2 API

During the first 3 weeks I worked more closely with the backend team to define the API that will be needed for me to send and get data from the DELA system. We decided to use an http server with GET and POST request. We came up with the API structure that evolved during the semester to its final version that is displayed in Figure 11. The white boxes are the JSON objects I need to send and the grey boxes the JSON objects I receive back.

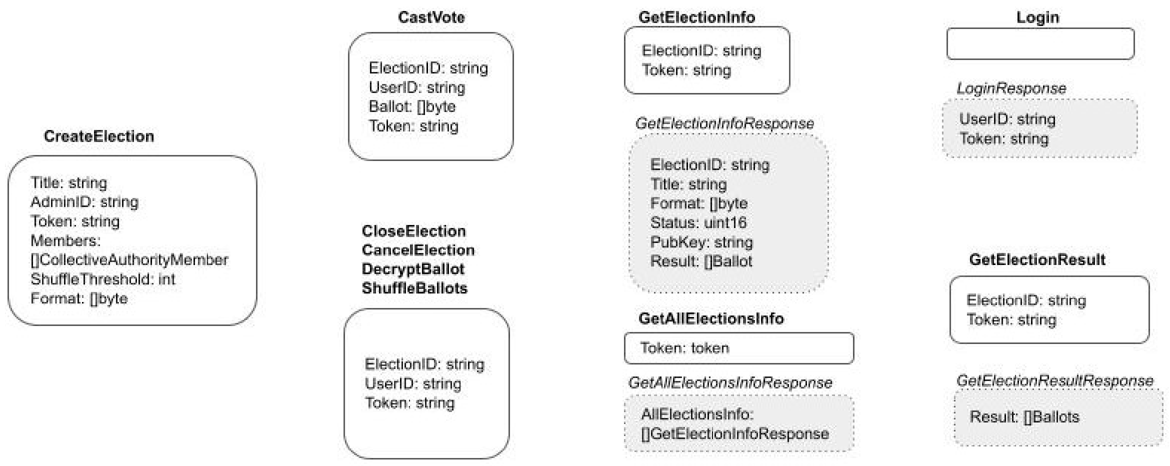


Figure 2: API requests and responses

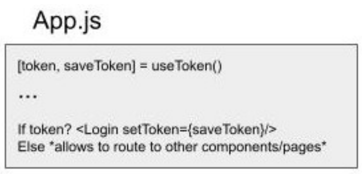
The field *Format* in the boxes CreateElection and GetElectionResponse corresponds to the election content. This was a last minute change by the backend team to replace a previous field name *Candidates* of type []string. During the entire semester, we defined the content of an election to be a title, an administrator ID, a list of candidates from which a voter will be able to select one

[8]https://github.com/dedis/student_21_d-voting_admin

(and only one), and an authentication token. API from before the last minute change and an example of the JSON object corresponding to the CreateElection can be found in Appendix A. This change will now give total freedom to the frontend to define what the content of an election is. However my code currently defines this *Format* field as `JSON.stringify('Candidates' : candidates)` with `candidates` being an array of strings.
The Ballot structure corresponds to the vote encrypted with the dkg public key as a string.

2.3 Authentication

The authentication used was also decided with the backend team with the idea that we would change it later for something more robust if we had time. We opted for a very simple one, an identifier and a token that a user gets from the login API endpoint. When obtained, those two parameters are stored in the browser's local storage and retrieved whenever they are needed as parameters in API calls.
The `Login` component is the login page that makes a call to the http server to get an identifier and a token. This component is displayed as long as the user didn't logged in. The file `App.js` uses the custom hook `useToken` which models the token state. If a token has been saved in the browser's session storage by the hook, a user can be routed to the pages of the webapp. Figure 3 illustrates how that works with pseudocode. One improvement of this project could be to


App.js

```
[token, saveToken] = useToken()
...

If token? <Login setToken={saveToken}/>
Else *allows to route to other components/pages*
```

Figure 3: Authentication summary

integrate Tequila authentication to replace the current one. Only `useToken.js`, `Login.js`, and the `if` condition in Figure 3 would need to be modified.

2.4 Encryption of the ballot

When an election is retrieved from the backend, one of its parameters is the election public key that comes from the DKG protocol of Dela. The vote needs to be encrypted with this key and I used the javascript kyber library [9]. The encryption is made by my function `encryptVote(vote, dkgKey, edCurve)` located in the file `VoteEncrypt.js`. It is crucial that the encryption of the vote is done on the front end side and not on the back end side to guarantee the anonymity of the vote. Indeed the connection to the http server is not secure so it is possible to observe the traffic and see the vote in clear if it is not encrypted before sending it to the server.
One important limitation of my encryption implementation is that the length of the vote to be encrypted cannot be > 29 bytes, otherwise the kyber library throws an error.
One possible solution for future improvement would be to modify my function `encryptVote` such that it would first generate a symmetric key, encrypt it on the elliptic curve with the dkg public key and then simply encrypt the vote with this new symmetric key. This would mean that the CastVote structure back in Figure 11 would have a new field that would hold the vote encrypted with the symmetric key and the Ballot would actually contain the encrypted symmetric key instead of the vote, along with the ballot encrypted with the symmetric key.
Another solution could also be to fragment the vote by batch of 29 bytes and then use the `encryptVote` function on each batch to encrypt. This solution would not be optimal if we would like to encrypt data way bigger than 29 bytes.

[9]https://github.com/dedis/cothority/tree/main/external/js/kyber

2.5 Modal

The `modal` folder contains two functional components, `Modal` and `ConfirmModal`. They replace the standard `alert(message)` and `window.confirm(message)` pop-ups. I wanted to be able to customize their appearance, their message. and the text displayed on their button(s). For the `ConfirmModal` I wanted to be able to pass as props some additional parameters. The signature of the component `ConfirmModal` is the following: `const ConfirmModal = (showModal, setShowModal, textModal, setUserConfirmedAction) => {...states and functions of the component...}` The `setUserConfirmedAction` function was needed to detect whether the user confirmed the operation (e.g. closing or canceling an election). The corresponding state `userConfirmedAction` is declared in the same file than where a `ConfirmModal` is used so that when `userConfirmedAction` is set to true with `setUserConfirmedAction`, the action can finally be triggered.

2.6 Implementations of tables to display data

The page that shows all the elections a user can take part in and the page that shows all the elections which have available results are similar in the sense that both filter elections based on their status (open status for the vote page and result available status for the result page) and both list only the title of the elections in a table (using material-ui [10]) with the title made clickable to redirect the user to a new url specific to the id of the election clicked. For these reasons I created a custom component `SimpleTable(statusToKeep,pathLink, textWhenData, textWhenNoData)` that fetches the election from the http server, filters them based on `statusToKeep` and displays them in a one column table or simply display `textWhenNoData` if there is no election to display. If in the future different types of election questions and formats are added, this component shouldn't need changes since it only displays the title of an election.

2.7 Testing

I used Jest testing framework since it came pre-installed with Create React App. All my tests can be found in the folder `src/components/_test_` .
Because of time constraints and my lack of experience writing tests, I wasn't able to test my entire code so I focused on what was the most important, which was to test that the encryption of the vote was made correctly. I also tested that `App.js` was rendering without crashing as well as some other components.

I was able to do more complex tests for my `Action.js` which is a component that display actions an administrator can do depending on the status of an election. For example, if the election is ongoing, it displays one close and one cancel button. If then the close button is clicked, a request to close the election is make with the API and if it is successful, a shuffle button is displayed. To test that those changes happened correctly, it meant that I had to automatically simulate a button click and then mock the API calls. I wasn't able to make it work with the Jest library alone and I had to use Enzyme[11] as well. This took me way more time that I would have thought so I didn't have time to do more tests.

2.8 Security

Since all the API calls are made to an http server, the connection is not secure. This means that if a malicious user is able to get the id and the token of another user (when the user is making the login request for example), he/she can then write them in his/her browser's local storage.

[10]https://material-ui.com/
[11]https://enzymejs.github.io/enzyme/

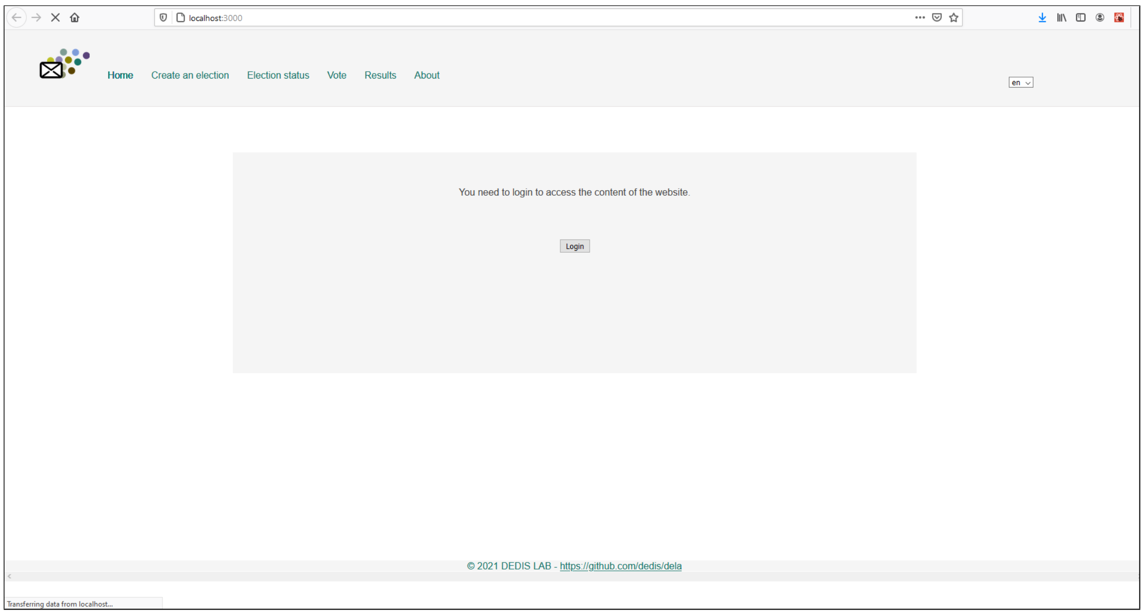
This would mean that it is possible to impersonate another voter when casting a vote since it retrieves the user id and the token from the browser's local storage.

3 Results

The current design of my web application doesn't restrict a user to access all the web pages once he/she is logged in. However apart from voting, other actions that can be made on an election (e.g. closing, canceling, shuffling, decrypting) won't work unless the user is the one who created the election. I chose a simple and neutral design, with neutral colors. The reasoning behind it is that it is an e-voting application and therefore it should give a "serious" impression on a user.

3.1 Navigation

The website is divided into six main pages that can be accessed through the navigation bar that is displayed at the top as shown in Figure 4. The navigation bar always appears on the screen and also offers the possibility to switch the language to French or English by clicking on the drop-down box at the top right of the screen. Clicking on the e-voting logo at the top left redirects to the Home page.



You need to login to access the content of the website.

© 2021 DEDIS LAB - https://github.com/dedis/dela

Figure 4: Display when the user is not logged in

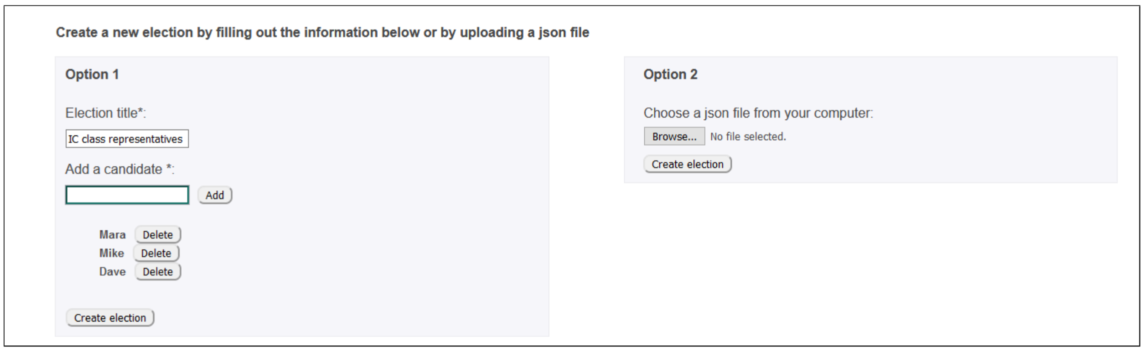
3.2 Creating an election

To create an election, the user can either fill out a form or simply upload a JSON file. Figure 5 shows the case of a user who filled out the form.

In terms of error prevention of the form, the user can not submit the election if the election name field has not been filled and it needs to have at least one candidate. It is also not possible to add two candidates with the same name and if the input field to add a candidate is not empty when submitting the election, it asks the user to either add the candidate or remove it from the input field.
For error prevention of the upload of the JSON file, it checks that the extension is indeed JSON, that it contains a Title field that is not empty and that the Candidates field is indeed an array of string. The object of the JSON file should contain the attribute of the CreateElection box in

Figure 11. However it does not work anymore because of the last minute change to the API and will need to be modified in the future.

If the election was successfully registered on a smart contract, a custom pop-up window tells the user so and the form fields are cleared out. If not, an error is also displayed to the user.


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

Option 1

Election title*: IC class representatives

Add a candidate *:

Mara
Mike
Dave

Option 2

Choose a json file from your computer: No file selected.

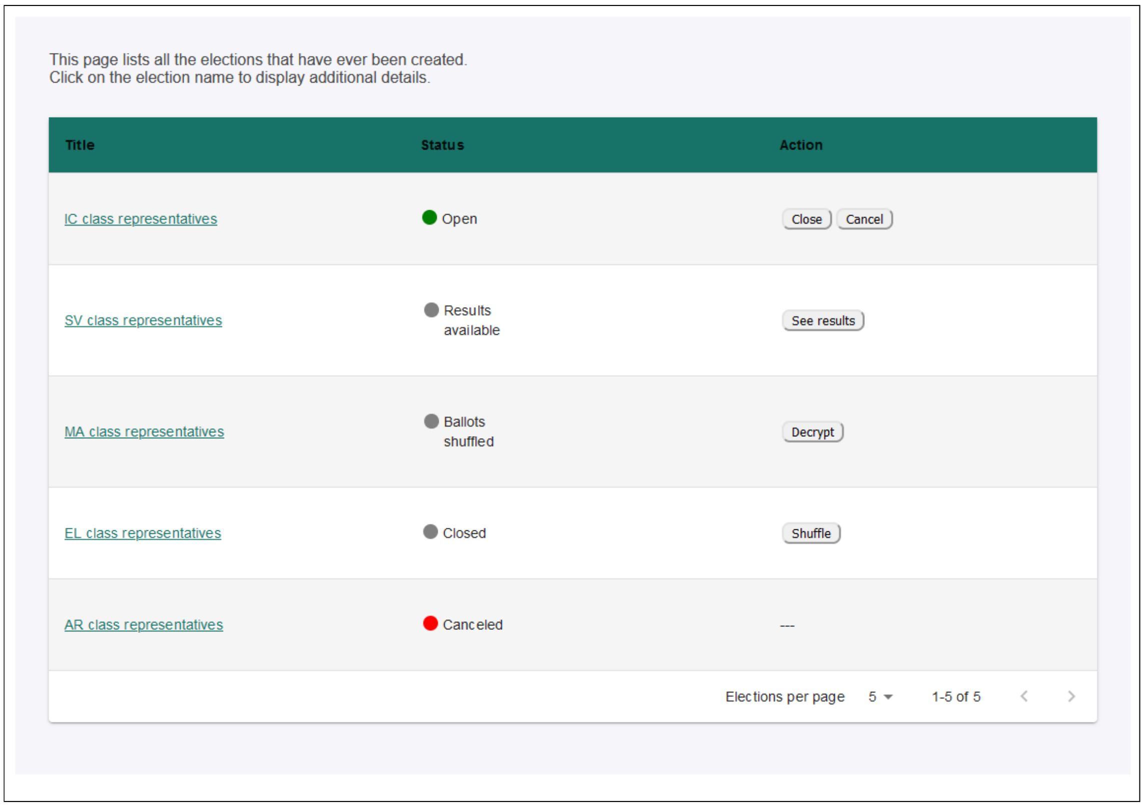
Figure 5: The user has filled out the form correctly and is ready to submit it

3.3 Election status

The goal of this page is for an administrator to have an overview of the state of all the elections that are registered on a smart contract. The elections are displayed in a paged table where one page of the table can display 5, 10, or 25 elections depending on what the user chooses.
To avoid displaying too much information, only the election title, its status and some actions that the administrator can trigger are displayed. All the actions except the one to see the result make an API call to trigger a change on the smart contract.
To see the details of an election, the user has to click on the title of the election which will create a new page with pathname 'elections/id_of_election'. If the results of an election are available, the action 'See results' will appear and clicking on it will direct the user to the same page than clicking on the election title. The results can be downloaded as a file with JSON extension on the specific election page..
As an example, Figure 6 displays all the type of status and actions that are possible and Figure 7 shows how results are displayed.
For the close and cancel action, a custom confirmation window makes sure that it was the user intended action. While all logged-in users can access this page and see the same display, not all actions will be successful. If a user clicks on an action button of an election he didn't create, the request (that contains his user id retrieved from his browser's local storage) will be sent to the http server. The http response will send back an error that is caught in the front-end implementation and display it in the custom modal.

3.4 Voting

Similarly to the election status page, all the on going elections are displayed in a table. The table has one column that only displays the title of the election. To cast a vote, the user has to click on the name to be redirected to a page specific to the id of the election where he will be able to cast its vote. An example of such a page is shown in Figure 8. Only one choice can be ticked per ballot and a user can vote as many times as he wants on an on-going election. However only the last vote will be taken into account once the election is closed. Every time the vote is successfully cast, the user sees a confirmation message on the screen.



This page lists all the elections that have ever been created.
Click on the election name to display additional details.

Title	Status	Action
IC class representatives	🟢 Open	Close Cancel
SV class representatives	● Results available	See results
MA class representatives	● Ballots shuffled	Decrypt
EL class representatives	● Closed	Shuffle
AR class representatives	🔴 Canceled	---

Figure 6: Example of a zoomed in display of all the elections that are stored on smart contracts

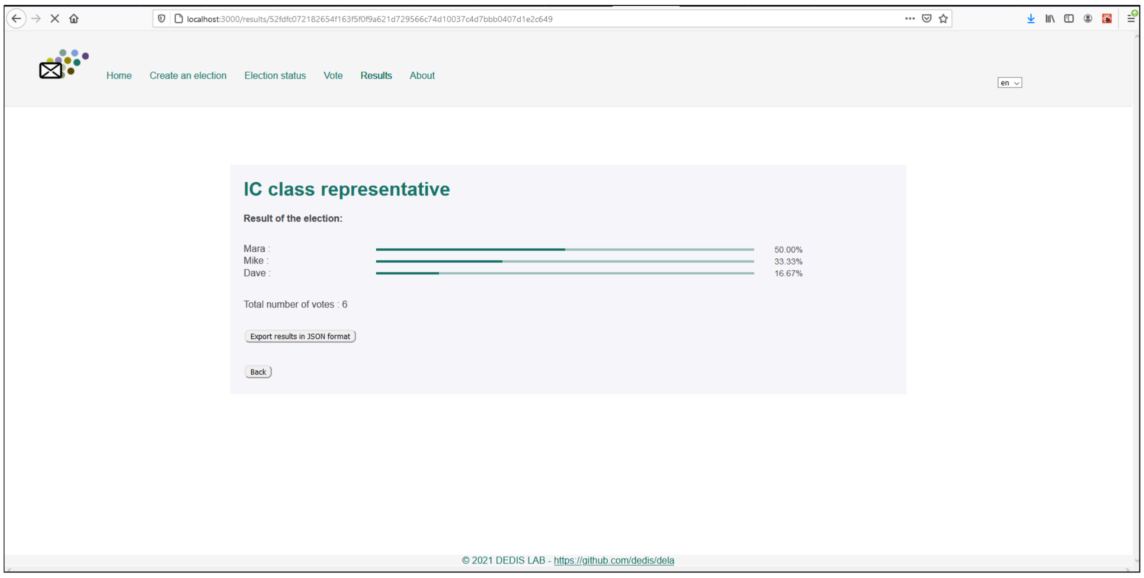
3.5 User experiment

During week 6 I asked two members of my family to give me some quick feedback about the UI. At that point it was still working with a mock API and not everything was done so I mostly asked for a feedback about color, layout, ... What came out of it was that the color I used for a table header was too aggressive and I had some layout issues.
During week 14 I performed another user experiment with the idea to really see how my web application was usable from a user perspective. Because of the sanitary situation at the time, I was only able to find 4 people from my immediate social circle, two in their 20s and two in their 50s. They tested my web application with Firefox browser. Before they started the study, I already created an election titled "Délégués de la section" on the system. I then asked them to perform two tasks :

1. Login, find the election titled "Délégués de la section" on the voting page, and vote for one of the candidates.

2. Create an election with 3 candidates, then vote for one of the candidates. *(At that point I would also cast a vote for the same election with another user id.)* Close the election, shuffle, decrypt and display the results before downloading them.

All my users were able to successfully complete the first task without any help on my part which is great news since that was simulating what a voter lambda should be able to achieve with my UI.
The second task caused some struggles to one of the user in their 50s which was to be expected since I wouldn't consider this person to be a computer person and the task asked was more of an administrator type of task. The first user pointed out during the election creation process (see Figure 5) that she was surprised of the behavior of the "add" button. She had expected the input field to add a candidate to go down instead of the added candidate being moved below the


localhost:3000/results/52fdfc072182654f163f5f0f9a621d729566c74d10037c4d7bbb0407d1e2c649
Home
Create an election
Election status
Vote
Results
About
en
Export results in JSON format
Back
© 2021 DEDIS LAB - https://github.com/dedis/dela

IC class representative

Result of the election:

Mara : 50.00%
Mike : 33.33%
Dave : 16.67%

Total number of votes : 6

Figure 7: Display of the results of an election

input field. The two other users however felt that it was fine and that the behavior was similar of how a todo list would add new items (which was what I was going for). When the first user was trying my UI, the status page (see Figure 6) did not have the third Action column. Instead the action buttons were simply displayed next to the status in the status column. I also didn't display the "See results" button because it seemed logical to me to simply click on the election title to display the election details which also meant its results. Based on her feedback and my own observations I decided to add the Action column and a button to see the results before the three other users tested my UI.
While witnessing my test users trying to see the results of an election, I realized that going through the Status page to be able to access the results was not the most logical flow for a voter and would definitely be problematic once some web pages will become admin-access only (which would be the case of the Status page), making it impossible for a voter to access the results. To solve that issue, I decided to add a Results page that would work a little bit like the Vote page. It displays a table with only the title of all the elections whose results are available and when the title is clicked, it opens a new page that display the specific results of the election chosen.
One other user noted that she was surprised by the fact that when asked by the confirmation window if she really wanted to close the election, the yes button was displayed on the right while the no button on the left. Another user also would have liked more flashy color.
Overall my test users didn't have too much trouble executing the two given tasks, they found the layout and the design to be simple but efficient.

4 Possible modifications/improvements

The main modification would definitely be to change the authentication to a Tequila one and define who would be an administrator. That would allow to really make some web pages not accessible by users who are not administrator because my current implementation doesn't really make a difference between an administrator and a simple voter.
I also wanted to add the possibility to create other type of forms, such as rank question and text question however I didn't have time left for this. That would probably require to create specific form components and add a drop-down menu for example in the Create an election page such that it would display one of the three types of form depending on the user choice. Thanks to the last minute change in the API with the *Format* field instead of the *Candidates* field, it now

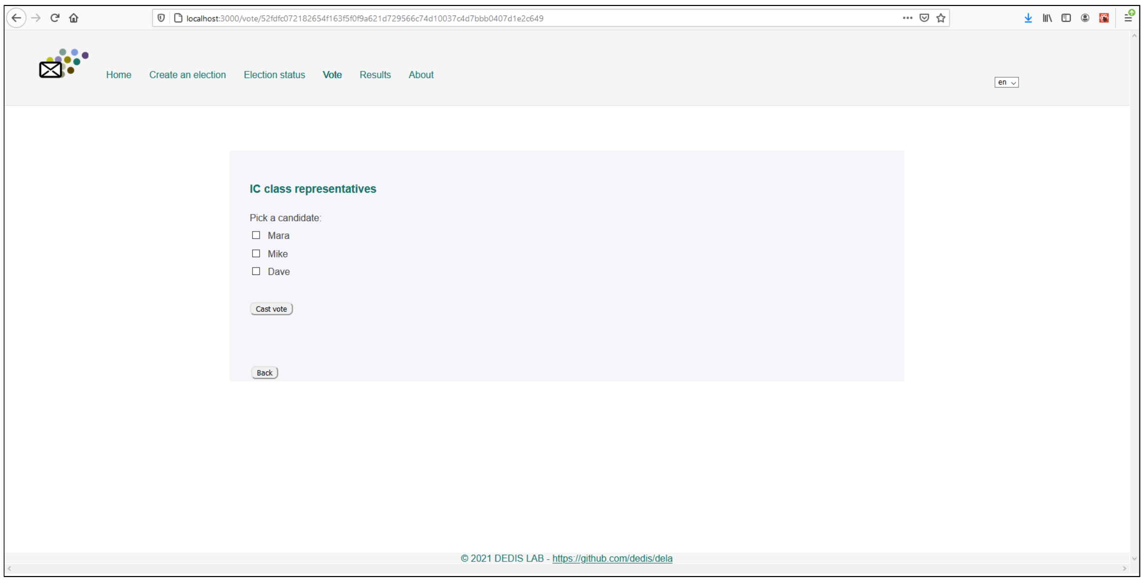


Figure 8: Example of a ballot

gives total freedom to the front end to define it without worrying about what happens in the back end. I think one way to go would be to simply nest different JSON objects, each containing the data of one question of the election.
That would also imply that `ElectionDetails.js`, `Ballot.js`, and `Result.js` will need to be modified in consequence to correctly display an election and its results.
Creating more complex elections by mixing multiple types of question, being able to vote for more than one candidate would also be nice. Setting a time where an election would automatically close would also be a nice feature.

As already stated in 2.4, the encryption of the ballot should be modified to be able to encrypt more than 29 bytes, especially if text questions are implemented.

In my current implementation, I am not making use of caching and I'm sure the number of time I make API calls to retrieve election data from the smart contract can be optimized. Furthermore a pagination mechanism should be added to avoid getting too many elections at the same time through the GetAllElections endpoint. At some point during the semester I was saving in the browser's storage the most recent vote for each election that a user took part in. The goal behind it was that if a user wanted to vote again in the same election, the last candidate she voted for would be displayed above the ballot. I removed it because using the browser's storage didn't seem like the best solution to me. I still think that would be a nice feature to have but saving it on a server would be better. When logging in, the user would get back from that server all her votes.

Another nice feature to add would be to be able to sort the way the elections are displayed in the status and the vote table. The way I coded `ElectionTable.js` and `BallotsGrid.js` shouldn't require too much modification in its structure since I used the react library Material-UI to construct the table [12] and they have an example where they added sorting to their table. For the table that displays the on-going elections a user can take part in, sorting it by date or by votations for which the user hasn't cast a vote yet would make sense.
For the status table, it would also make sense to be able to sort it by creation date or by status. Furthermore being able for example to select several elections and close them at the same time could be practical from an administrator's point of view.

[12] https://material-ui.com/components/tables/

The explanations about how an election work in the About page are very short and not detailed and that can definitely be improved by describing in much more details how the system work and adding some figures to visualise it. Adding tutorial and a FAQ section could also be a possibility.

5 Conclusion

The goal of this project was to implement a front-end to an e-voting system on Dela. While my implementation doesn't offer a lot of freedom in terms of the type of election one can create, it definitely works with a simple election where a user needs to choose one candidate among others. Based on the feedback collected during my user experiment, I know that my interface is usable by lambda users and I feel confident that with some more work it could be used for future EPFL elections.

5.1 Personal Experience

When I started this project, I had no experience in front end, nor in Javascript, html or CSS. My goal was really to learn something new and to have some freedom in terms on how I wanted to implement it and its architecture.
What went well was my work pace. I was able to implement what I had planned almost every week and every challenge I encountered along the way were very stimulating to me. For example, changing the language on the website at any time, or how it was possible to pass data between back end and front end and also how to deal with loading time and error when fetching data.

While at first I really wanted to implement most things from scratch and use as few libraries as possible, I realized that would not be possible if I wanted to have time to have a working front end at the end. I feel like I postponed really testing my code til mid-semester and at that time my code was more complex to test which felt overwhelming. That set me back a little bit in my timeline and I was not able to implement different type of election questions like I wanted.

A Annex

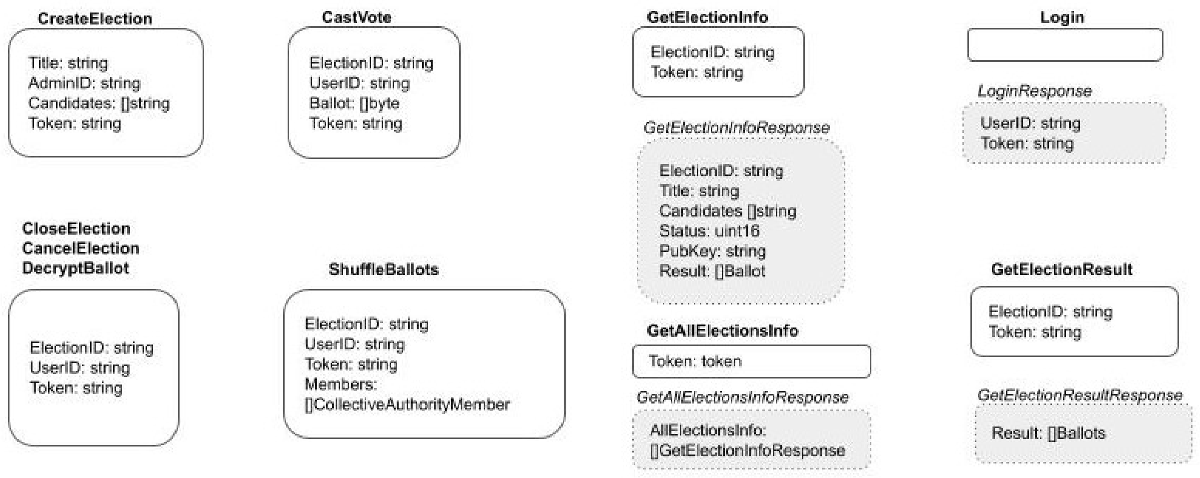
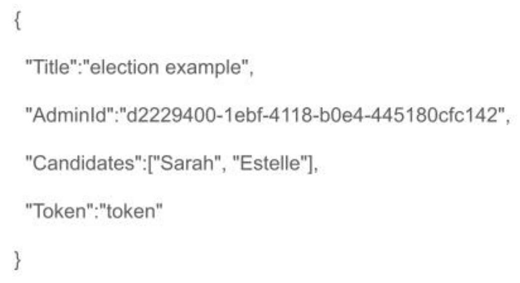
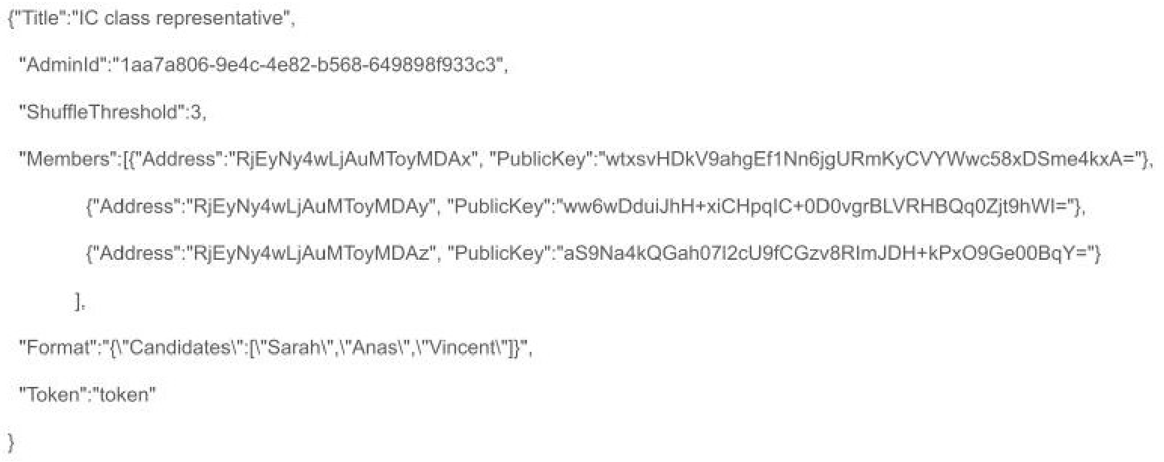


Figure 9: API structure until June 9th



```
{
 "Title":"election example",
 "AdminId":"d2229400-1ebf-4118-b0e4-445180cfc142",
 "Candidates":["Sarah", "Estelle"],
 "Token":"token"
}
```

Figure 10: Example of a JSON object to create an election before the new update



```
{"Title":"IC class representative",
  "AdminId":"1aa7a806-9e4c-4e82-b568-649898f933c3",
  "ShuffleThreshold":3,
  "Members":[{"Address":"RjEyNy4wLjAuMToyMDAx", "PublicKey":"wtxsvHDkV9ahgEf1Nn6jgURmKyCVYWwc58xDSme4kxA="},
            {"Address":"RjEyNy4wLjAuMToyMDAy", "PublicKey":"ww6wDduiJhH+xiCHpqIC+0D0vgrBLVRHBQq0Zjt9hWI="},
            {"Address":"RjEyNy4wLjAuMToyMDAz", "PublicKey":"aS9Na4kQGah07I2cU9fCGzv8RImJDH+kPxO9Ge00BqY="}
          ],
  "Format":"{\"Candidates\":[\"Sarah\",\"Anas\",\"Vincent\"]}",
  "Token":"token"
}
```

Figure 11: Example of the new JSON object to create an election