Università di Trento

Master degree

Information Engineering and Computer Science Department

Blockchain project

DelegateXpress

Authors: **Marco Xausa - Gabriele Abbate**

Trento, 11/2023

Contents

List of Figures

1 Business analysis

1.1 The problem

While doing the interviews to understand what problem our group could solve with the help of the blockchain, we came across a very common yet cumbersome practice many people face everyday: the tedious process of creating, signing, and managing delegations. Both companies and customers face the need to delegate, yet the current paper-based approach requires in-person signatures, resulting in a significant waste of time. For a delegation to become effective, parties must wait for the paperwork to be finished, forcing a delay in the termination of the task. The way delegations are typically managed has not changed in decades, and is affected by some problems. They do not happen frequently, but if they do, they can be a big problem for the users. Among all, this delegation process causes particular difficulties for customers managing unforeseen events. If a customer cannot be physically present to sign forms due to geographic separation, the situation becomes extremely inconvenient, since the delegation will not be valid until the sign is done.

On the other hand, for companies, delegations represent a necessary but secondary activity. While customer satisfaction and efficient operations are vital, managing delegations distracts from the core business. Handling, checking, and updating delegations for a large number of users and for different services can become not trivial for big companies, especially if they are done both digitally and on paper way. Paper delegations can in some cases optimize workflows, since they are easy to compile and most of people fill them quite easily, but yet when there are many delegations for many different user, the amount of paper can become hard to maintain.

In brief, the manual delegation process hampers productivity for both clients and companies. Our solution aims to solve these inefficiencies by streamlining the delegation workflow. This will save time and resources while enabling flexibility for off-site customers and internal teams. By reducing delegation paperwork, companies can better focus on customer needs and core operations.

1.2 DelegateXpress

Based on the above problems, we have developed a proposal which can overcome these issues thanks to the adoption of blockchain technology. The idea is to leverage the latter to keep track of the delegations, making them easy to manage, verify, and modify. All of this is in a framework that is law-compliant.

The solution is a Web application (or better, a Decentralized application), which relies on blockchain to store, check, and delete delegations. In this way, it is possible to guarantee the reliability of the data and its immutability. The service will be available 24/7 from everywhere, erasing the need to sign any document and allowing users to delegate whenever they want.

The user base will be composed of institutions (mostly banks, municipal offices, and postal offices) and their users (employees and customers) which will be able to delegate other users for services of a certain company. The delegation will be available online on a platform where the delegator will only need the identifier of another user to be able to delegate them for a certain activity in their place.

In summary, our business will be based on 3 components: a blockchain to save the smart contracts about delegations, a database to save institutions and users of the network and a web page to make institutions/users sign and manage delegations.

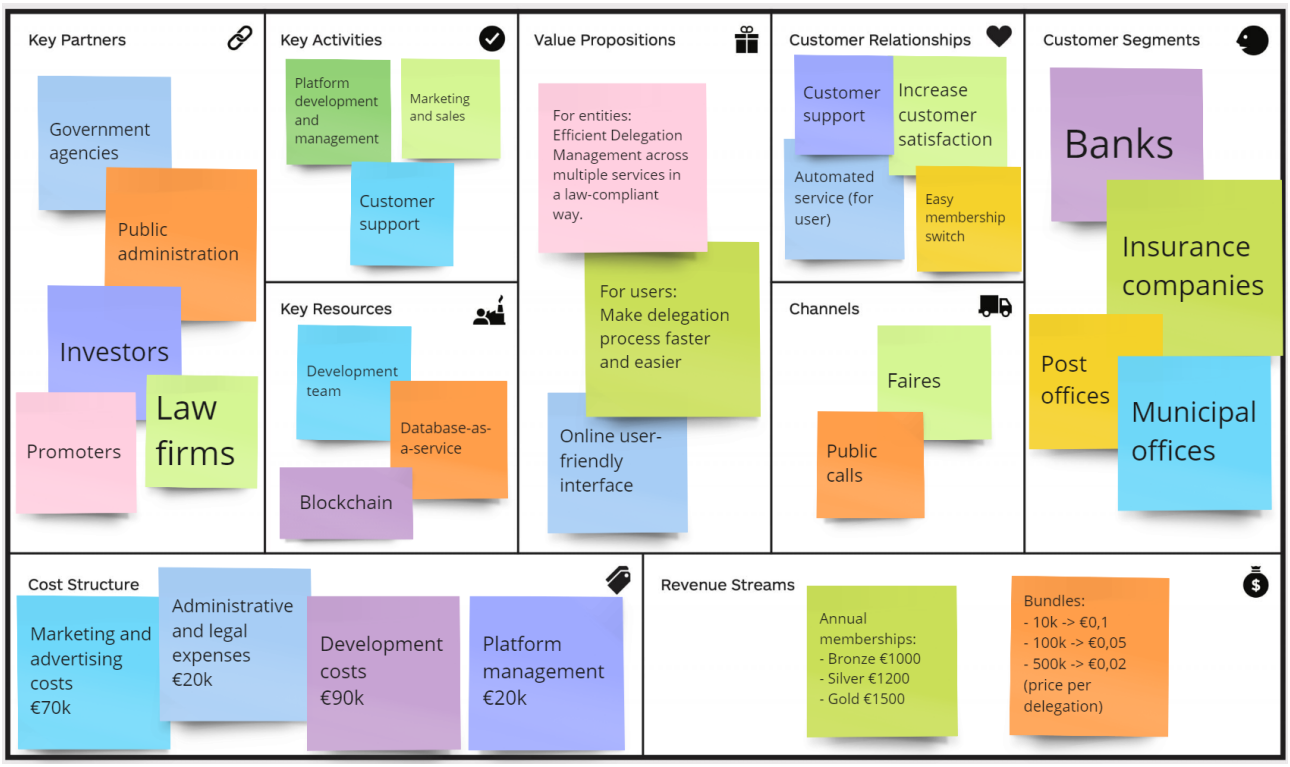


Figure 1: Business Model Canvas

1.3 Market analysis

Regarding the business side, we decided to start our study with a market analysis to assess our business's target market and the competitive landscape. In this phase we consider the framing of our future customers to be crucial, that is why the next few pages will address each of the business model sections in a clear and detailed way.

1.3.1 Business Model Canvas

- **Customer segments**: Our business focuses on serving a diverse range of customer segments, including banks, insurance companies, post offices, and municipal offices. These entities are chosen because they share a common need for effective delegation management and data integrity. Banks require secure and streamlined delegation processes for financial transactions, while insurance companies need to manage policyholder information. Post offices and municipal offices often handle various administrative tasks that involve delegating authority and ensuring accurate data handling.

- **Customer relationship**: Our business is committed to nurturing strong and lasting customer relationships through several strategies. We prioritize exceptional customer support to promptly address any inquiries or concerns. We also aim to enhance customer satisfaction by offering reliable, user-friendly and efficient software solutions. Our platform's automation features help clients save time and effort, while our support for easy membership switches ensures flexibility for evolving needs.

- **Channels**: In order to connect with our target customers, we leverage effective channels commonly used in our industry. Participating in fairs and public calls enables us to showcase our software platform directly to potential customers. These interactions create

opportunities for direct engagement and meaningful conversations, fostering trust and familiarity.

- **Value proposition**: Our software platform's value proposition is for both the entities using the platform and the end users. For the entities, our platform streamlines delegation management, increasing operational efficiency and data integrity and ensuring a law-compliant structure if needed. For users, the online user-friendly interface simplifies and accelerates the delegation process, improving their overall experience and making it available at any time and any place.

- **Key Activities**: The core activities of our business encompass various crucial functions. Development and maintenance of the platform is crucial to ensure good service. We prioritize user interface development and maintenance to ensure a nice and intuitive experience. Other important core activities are related to marketing and sales, and the customer support.

- **Key Resources**: To execute our vision, we rely on key resources. A strong development team is fundamental for building and maintaining the platform and the user interface. Another important resource is the database-as-a-service solutions, to boost our scalability, reliability, and cost-effectiveness. Last but not least, the blockchain, which is the core value of our product, and a resource without whose our business would make no sense.

- **Key Partners**: Our success is linked with key partners who contribute pivotal support. Government agencies and public authorities provide regulatory guidance, ensuring compliance and credibility. Investors help us to grow by providing essential funding and resources. Collaborations with law firms and promoters facilitate legal and promotional support, respectively, enhancing our startup's viability.

- **Cost Structure**: Our initial funding is allocated across key areas critical for our startup's success. Development costs, encompassing salaries, licenses, tools and infrastructure setup, form the largest chunk. Marketing and advertising expenses are prioritized to promote our platform through various channels. Administrative and legal expenses cover essential office-related needs, while maintenance and support costs ensure our platform's continuous functionality, security, and updates.

- **Revenue streams**: We generate revenue through two primary streams. In the first place, we will offer three different annual subscriptions:

  - Bronze: gives the institution access to the platform and allows it to use the platform with no personalization and a basic customer care service. Cost per year: 1000€;
  - Silver: gives access to the platform, but allows customization based on the business need, also with the benefit of advanced customer support. Cost per year: 1200€;
  - Gold: Gives all the benefits of the Silver subscription, but allows the company to integrate our solution on their website through our system APIs. Cost per year: 1500€;

  The second stream comes from delegation bundles, where the pricing per delegation varies based on volume, catering to the diverse needs of our customers. The bundles are:

  - 10k delegations for €0,1 each;
  - 100k delegations for €0,05 each;

– 500k delegations for €0,02 each;

These revenue streams ensure our financial sustainability and growth.

1.3.2 Initial investment

To estimate the initial investment needed, we need to consider all the activities, resources, and human capital required to build a first valuable product. The area of main interest will be for sure development. This is what most of our activities will be focused on. To start the business, also a considerable part of the investment must be allocated to marketing and promotion, which is vital to let our customers get to know our company and our solution. Moreover, expenses such as salaries for the development team, platform maintenance costs, server hosting fees, and legal costs must be taken into account even though they will not impact the cost structure as they will in the future. A reasonable amount of money we believe can allow us to kick off with this startup is €300.000.

1.4 Financial projection

We performed market research and studied industry trends in order to be able to project the future of the startup for the next three years, the moment in which we aim to reach the breakeven point and become profitable. To complete this process, we made some assumptions about the number of delegations needed by companies, the number of bundles we will be selling to our customers, and our capacity to attract new customers. The path we aim to follow with the company for the next year is the following:

- **First year**
  For the first year, most of the effort will be focused on the development of the application to have a final product. In the first stage, only the founders of the startup will be working on it remotely, minimizing the initial costs. Also, there is the need to setup and manage the platform which will host the solution. Lastly, the first marketing campaigns will take place to have potential customers once the product is ready. Assuming we are a Team of three developers, we believe about 200.000€ will be needed to pursue this first milestone. This expense represents the entire negative balance of the first year.

- **Second year**
  By the beginning of the second year, we will have already started selling our solutions. If we manage to have 30 bronze, 15 silver, and 5 gold subscriptions, we will be generating €57.000 from yearly subscriptions. If these companies buy a total of 50 delegations bundle (1 each), of which 40 for 10k delegations and 10 for 100k delegations. This would generate a revenue of 90.000€, for a total income of 147.000€. However, the expenses this year will be more than duplicated: the customer assistance service must be assured and considering operational and marketing costs we will reach an outcome of 340.000€.

- **Third year**
  In the third year we belive we will be reaching enough customers to cross the breakeven point and become profitable. With 43 bronze subscriptions, 22 silver, and 7 gold, we will be generating 81.600€ from subscriptions. Assuming we sell a total of 75 bundles for 10k delegations, 35 bundles for 100k delegations, and 7 bundles for 500k delegations, the income from bundles will be 320.000€, for a total yearly revenue of 401.600€. The total expenses for salaries, marketing, and operational costs are about €400k, hence allowing us to reach the profit.

2 Development workflow

The project development has evolved throughout the process. In the first phase, a subdivision of the program functionalities was made, leading to different inefficiency in the workload subdivision and a non-optimal delivery.

As the team decided to resubmit the project, a completely new approach was adopted. We decided to follow a kanban methodology, defining tasks that members could take charge of, and allowing the other members to review the work done by the others before considering the task as "done". The definition of the to-do tasks was the first point. The focus was on outlining the steps to use to improve the previous delivery both from the code and the business point of view without going too much into detail but rather describing the final objective of each task. The final list of tasks describes the optimization problems we wanted to tackle and the lacking points we wanted to improve from the first delivery of the project.

The screenshot of the kanban board can be seen in Figure 2

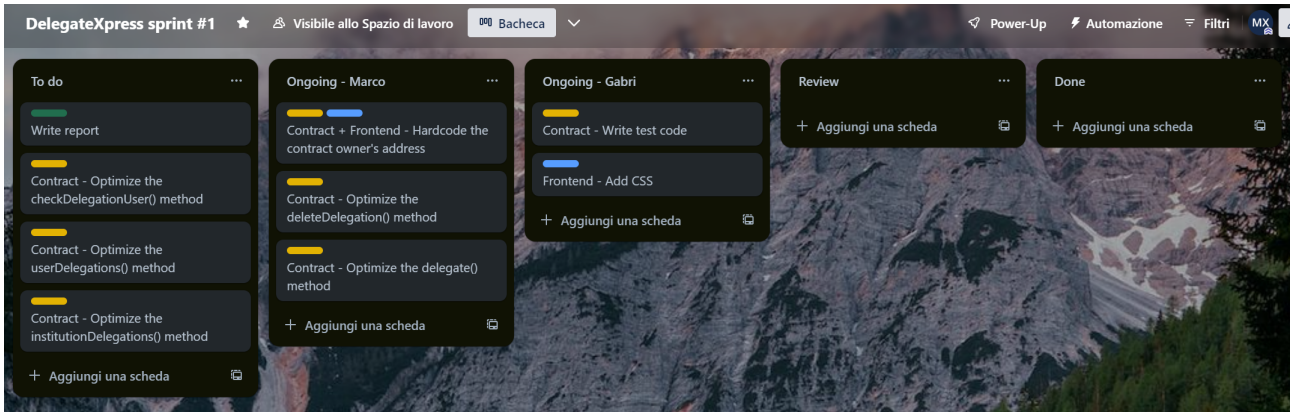


Figure 2: Kanban board

From here, each member could start working on a task by dragging it to their respective ongoing column. Once the task was finished it could be moved to the review area, in order to let the member review the code and confirm the task was correctly fulfilled.

This Kanban board has been iterated two times, with the same tasks. The reason for this is not to optimize again the same contract, but rather to have different versions of the contract with the same functionalities. As a result, three different versions are now present in the project:

- **Delega.sol**, the first version of the contract, which will be used to evaluate the improvement in performance;
- **Delega2.sol**, the first optimization approach approach, with a structured representation of the delegations;
- **Delega3.sol and DelegationStorage3.sol**, The second optimization approach, is implemented with an upgradability pattern splitting the business logic and the data.

Since we did not know prior which approach would have been the best one, and which data structure would have returned the best performances, the possibility of testing different contracts gives us the possibility to compare the costs and understand which one is the best solution.

3 Structure of the application

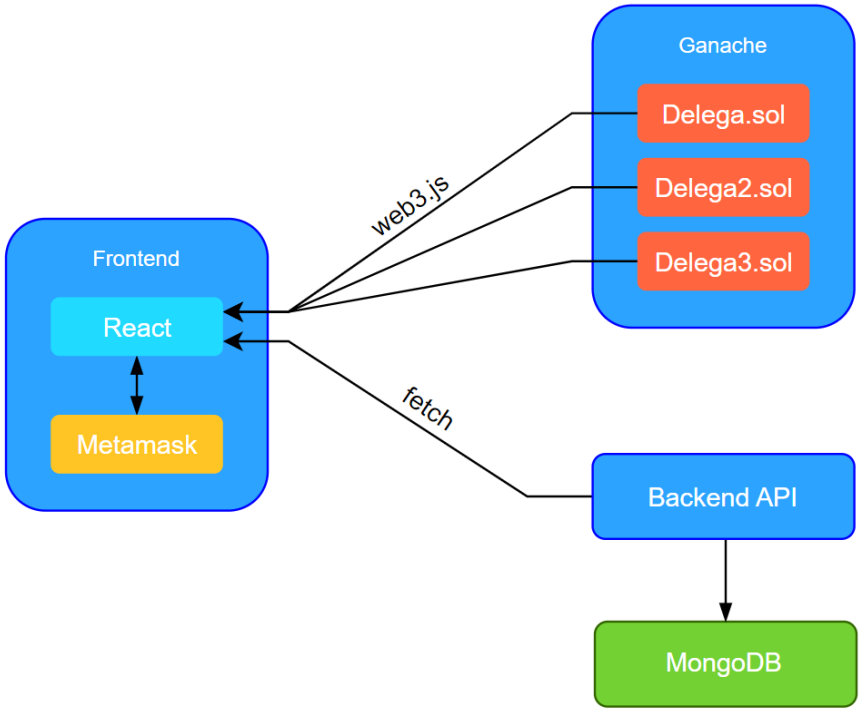


Figure 3: Components of the application

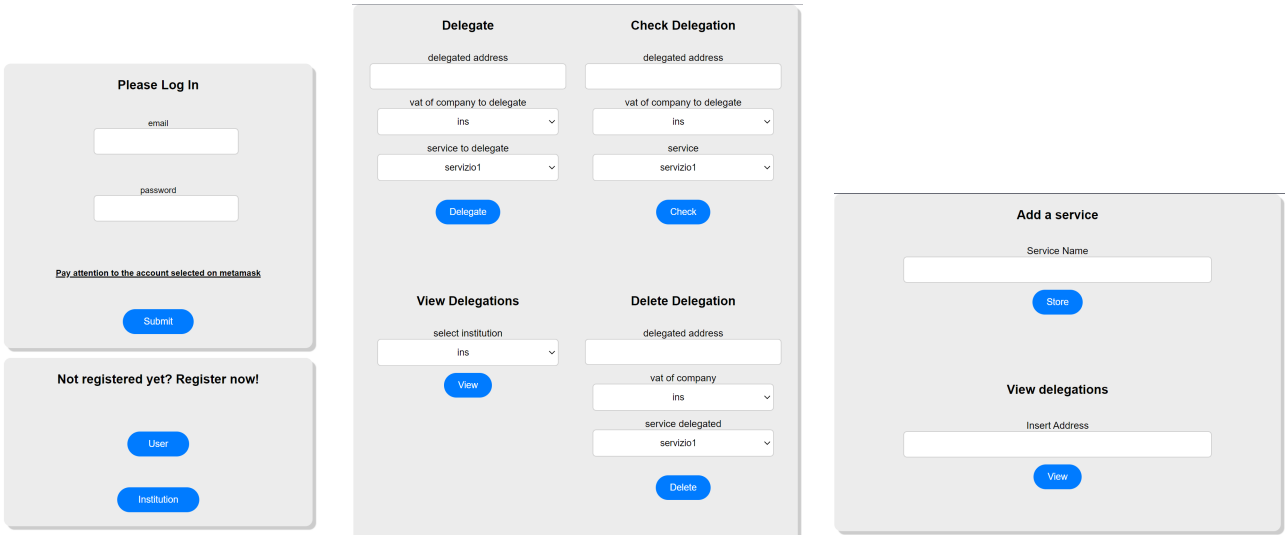
The application is split into four main components as represented in Figure 3:

- Frontend, built using React.js, utilizes Metamask to interact with the contracts instantiated on Ganache;
- Ganache, which is the virtualized Ethereum blockchain with all the contracts;
- Backend APIs, developed through Express.js, they offer some authentication and authorization functionalities as well as the mechanism to make the service accessible only through our online platform. Different versions exist
- MongoDB, which is the database supporting all the functionalities of the API.

3.1 Frontend

As described, the frontend has been developed with React.js, starting from the React Truffle Box. The code is split into different parts:

- Components, which are the pieces of interface the user can interact with;
- Contexts, which take care of retrieving the contract and making it available in every component;
- Contract ABI's, which is a folder containing all the json files produced after the deployment of the contract using truffle



(a) Initial page (b) User page (c) Institution page

Figure 4: Frontend screenshots

The initial page presents two different possibilities: login as an already existing user/institution or the possibility to register to the platform. The usage of the platform starts with the registration. The user can choose how they want to register by clicking the institution or user button. When the button is clicked, it will display the specific form for the registration (4a).

Once registered, both institutions and users can login from the same form. After the authentication, the application will display the page with the account's related functionalities. The user page (4b) has four different areas corresponding to the actions a user can do:

- **Delegate a new user**, which takes as input the address of a registered user, the institution, and the service and stores a new delegation on the blockchain;
- **Check Delegation**, that checks if an address is delegated for a specific institution and service;
- **View the delegations** they authorized for an institution;
- **Revoke** a delegation;

The institution page (4c), gives access to two functionalities:

- **Add a service**, which lets the institution add a service to be delegated by the users;
- **View delegations**, which allows the institution to check all the delegations made by a user for every service;

The focus of the rework was to make the interface more user-friendly, with the objective one of the value proposition points. For this reason, a CSS stylesheet has been added to each component.

3.2 APIs

The backend has been implemented with Node.js, leveraging the Express.js framework. It offers a series of endpoints to manage authorization, authentication, and encryption-decryption of services. The APIs have seen some reworks and are split into 2 different versions, recognized

by the `/v1` and the `/v2` prefixes in the url. Some of the most important endpoints in the first version are:

- `/registeruser` and `/registerinstitution` to add new accounts on the platform. Whenever an institution is created, a symmetric key is also created to encrypt data;
- `/authenticate` used to validate the credentials of a user or an institution;
- `/encode` used to encrypt the string stored in the blockchain, making the service utilizable only with our Dapp. The encrypted string is formed by the concatenation of the institution vat, the tax code of the user, and the service, using the character — as a separator and encrypted with the institution's symmetric key.
- `/decode` used to decrypt the string returned by the blockchain and return clear data to the frontend;

Besides these, some others have been implemented to support the visualization of the data in the application.

Because of the changes made in the `Delega3.sol` contract, some modifications in the business logic were needed with the `/v2` version. Since only hashed strings are stored on the blockchain, the backend no longer needs to encrypt and decrypt data but rather to keep track of a hash-service string association, making the service strings management much different. For this reason the endpoints `/registerinstitution`, `/addservice`, and `/decode` were modified accordingly. This approach still makes our service accessible only by means of our APIs and the web application.

4 The contracts

As mentioned above the project encompasses multiple versions of the contract. The first version, `Delega.sol`, represents the ground for improvement and performance evaluation in terms of gas fees. Both `Delega2.sol` and `Delega3.sol` use new different approaches and data structures to represent the delegations to achieve the objective. However, some functionalities remain unvaried among all the versions. For example, two mappings indicate which institutions and users are allowed to interact with the contract's methods (after the registration process). The contract owner is the only method allowed to call these methods, even though this functionality has been fully implemented for both `Delega2.sol` and `Delega3.sol` together with the frontend.

The functions for the registration of a new account are the same in all the versions:

- `function addUser(address user) public` →Used by the contract owner, is invoked when a new user subscribes to the platform to enable them to invoke the methods;
- `function addInstitution(address institution) public` →Used by the contract owner, is invoked when a new institution subscribes to the platform to enable it to invoke the methods;

Some functions have changed between the first and the second version because of changes , but remain unchanged in the third, like:

- `function compareStrings(string memory a, string memory b) internal pure returns (bool)` →Used to compare to strings, returns true if they are equal;
- `function addService(string memory service) public` →Used by an institution to add a delegable service to the platform;
- `function checkService(address institution, string memory service) public view returns (bool)` →Used to check if a string corresponds to a delegable service;

while some others have been changed or implemented only in the second and third versions:

- `function isAuthorizedUser(address user) public view returns (bool)` →Used check if an address belongs to an authorized user or not;
- `function isAuthorizedInstitution(address institution) public view returns (bool)` →Used to check if an address belongs to an authorized institution or not;
- `function hash(string memory a) internal pure returns(bytes32)` →Used to ease the calculation and increase code readability;

Each contract uses its own data structure, and the study of how each approach impacts the core methods of the contracts is done below.

4.1 Delega.sol

This version of the contract uses the simplest approach possible. A delegation is only described by a delegated, an institution, and an array of services for which the address is delegated. A mapping `delegations` gives access to the array with all the delegations made by a user.

```
    struct Delegation{      //a delegation is made by the address of the
   delegated person and the service for which it is delegated
        address delegated;
        address institution;
        string[] services;
    }

    //each address represents a user and their delegations
    mapping(address => Delegation[]) delegations;
```

Utilizing such a simple approach leads to inefficiencies in the code, which are reflected in higher gas fees which will increase as the dimension of the data grows. The logic behind the most important methods is the following:

- `function delegate(address delegated, address institution, string memory service) public`

  The method creates a new delegation. The msg.sender, the delegated and the institutions have to be authorized accounts. If the require clauses are successfully overcome, it is immediately checked if the msg.sender already has delegations or not. If delegations are already present, a delegation with that delegated and that institution is searched, and if found, the service is attached to that. Otherwise, a new delegation object is created and pushed in the delegations array;

- `function checkDelegationUser( address delegated, address institution, string memory service) public view returns(bool)`

  Check if a delegation for that delegated address, institution, and service exists. If any of the msg.sender, delegated, or institution is not an authenticated user returns false. Otherwise, a while cycle looks for the service among the delegations, and if found returns true.

- `function userDelegations(address institution) public view returns (Delegation[] memory)`

  Retrieves for all the delegations of a user for a specific institution. It takes the count of delegations for that institution with a for cycle, creates a memory array with that dimension, and then uses another cycle to populate the array and return it to the caller

- `function institutionDelegations(address user) public view returns (Delegation[] memory)`

  Returns to an institution all the delegations of a user for services related to the institution. Works similarly to the above methods, use a for cycle to get the count, create a memory array of delegations, and then populate it with another for cycle and return it to the caller.

- `function deleteDelegation(address delegated, address institution, string memory service) public`

  Used to remove a delegation. It controls if msg,sender, delegated and institution are all addresses allowed. It also checks if the checkDelegationUser function is true. If the require clauses are met, the method calculates the index of the delegation related to that institution and the index of the service with two nested while loops. When found, it can proceed with the deletion. Depending on whether the delegation is the last or not and the service related to the delegation is the last or not it is performed differently.

4.2 Delega2.sol

The first optimizations tackled with this contract were the reduction of loops and the usage of strings throughout the code. In order to do that, a more sophisticated data structure has been designed, which also encompassed information on how to access data quickly with no iterations. As a result, a delegation is no longer a single structure, but rather a hierarchy of mappings and structures, each representing a different stakeholder of the delegation problem:

```
1   struct Services{        //represents the services of an institution
    delegated to every single delegated
2
3       mapping(bytes32=>bool) serviceIsPresent;
4       mapping(bytes32=>uint) serviceIndex;
5       string[] services;
6       address delegated;
7   }
8
9   struct Delegateds{      //represents the delegated for every single
    institution
10
11      mapping(address => Services) delegateds;
12      mapping(address=>bool) addressIsPresent;
13      mapping(address=>uint) delegatedIndex;
14
15      address[] delegatedAddresses;
16      uint numberOfDelegations;
17  }
18
19  //users' delegations
20  mapping(address=>mapping(address=>Delegateds))  users;
```

With such a structure, using the `users` mapping, it is possible to access all the delegations of a single user for a given institution through the `user's address - institution's address` couple. This returns a structure with all the delegated addresses is returned which itself contains a mapping with all the services for which an address is delegated (Figure 5).

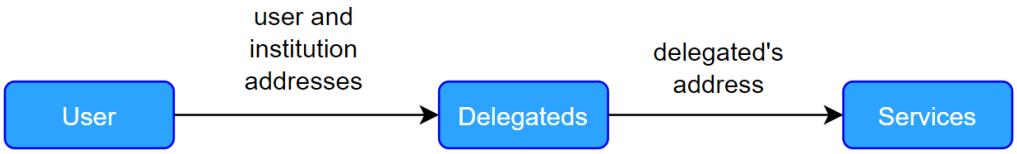


Figure 5: Delega2.sol - Flow to access a delegation

Both the `Services` and the `Delegateds` structures contain support variables, such as `servceIndex` or `serviceIsPresent`, whose purpose is to make insertion, deletion, and verification faster and avoid the needs for loops. These changes help make the functions more straightforward to implement and code easier to interpret. The way this data structure impacts the most important methods is the following:

- `function delegate(address delegated, address institution, string memory service) public`

  This method takes care of recording a new delegation in the blockchain, focusing on correctly updating also the support variables accordingly;

- `function checkDelegationUser( address delegated, address institution, string memory service) public view returns(bool)`

The method checks if the hash of the service is present among the delegated services thanks to the serviceIsPresent mapping, and returns the value returned by the latter;

- `function userDelegations(address institution) public view returns (returnValue[] memory)`

  This method iterates through all the delegated addresses for an institution, adding it to the returning struct, and for each address retrieves and returns all the delegated services;

- `function institutionDelegations(address user) public view returns (returnValue[] memory)`

  This function is similar to the one above, but instead, it iterates over all the delegate addresses for different users for the same institution and again, returns an array composed by address and array of delegated services;

- `function deleteDelegation(address delegated, address institution, string memory service) public`

  This function deletes the delegation if it exists. It checks whether it's the last delegation for the address or not. In both cases a specific deletion method is invoked which properly erases the variables and fixes the support variables;

Concerning the frontend, in this contract, the method signatures remain unchanged, allowing the already existing version of the code to work seamlessly with the contract. The APIs still provide methods for authentication, encoding, and decoding the services in order to make the contract's data utilizable through our Dapp.

4.3 Delega3.sol

For this version, the goals were still the same as for the previous one, but with a focus on reducing the support variables in order to reduce the number of read and write operations during the delegation and revocation process, allowing the presence of loops in `view` functions and reducing the number of support variables in the structs. The data structure is encoded by the following:

```
1    struct Service{
2        mapping(address=>bool) isDelegated;
3        address[] delegatedAddresses;
4        uint256 delegationAmount;
5    }
6
7    struct Institution{
8        mapping(bytes32=>Service) services;
9        bytes32[] allServices;
10       mapping(bytes32=>bool) servicePresent;
11   }
12
13   //users' delegations
14   mapping( address=> mapping(address=>Institution ) )  users;
```

With this structure it is still possible to access all the delegations of a single user for a given institution using the `user's address - institution's address` by means of the `users` mapping. Once obtained information about the institution, we can access all the services the user delegated for that institution with the `services` mapping using the hash of the service. With the `isDelegated` mapping, we can quickly check if an address is delegated for that service or not. The logical flow is represented in Figure 6.

Figure 6: Delega3.sol - Flow to access a delegation

In this version, there are no strings stored in the blockchain as state variables. All the services are hashed, and the APIs are used to ”decode” them. Indeed the backend encompasses new endpoints and structures to manage this workflow.

The list of the methods that changed is the following:

- `function delegate(address delegated, address institution, string memory service) public`

  The function set the `isDelegated` mapping value to true, and add the delegated's address to the `delegatedAddresses` array. Then increments the `delegationAmount` value. If the service was not already delegated, it is also stored to the `allServices` array , setting the `serviceIsPresent` mapping value to true;

- `function revoke(address delegated, address institution, string memory service) public`

  The function simply set the `isDelegated` mapping value for the specific delegated and institution to false and decreases the delegation amount, if the delegation was present;

- `function checkDelegationUser( address delegated, address institution, string memory service) public view returns(bool)`

  the method returns the value of the `isDelegated` mapping for the delegated and institution passed as parameters;

- `function userDelegations(address institution) public returns (returnValue[] memory)`

  Can be invoked only in `msg.sender` is an authorized user. The function calculates the length of the output array, then instantiate the array. It is filled with a nested for loop that iterate over every service and every delegated user and add them to the return array and returns it to the caller;

- `function institutionDelegations(address user) public view returns (returnValue[] memory)`

  Can be invoked only in `msg.sender` is an authorized institution. The function calculates the length of the output array, then instantiate the array. It is filled with a nested for loop that iterate over every service and every delegated user and add them to the return array and returns it to the caller;

4.3.1 DelegationStorage3.sol

For this version, the data separation upgradability pattern has been implemented. This choice was taken because of the multiple changes made to the contract's methods. Splitting the data and the business logic in such an application allows for reworking or adding features without losing already existing delegations. Hence, `Delega3.sol` derives from `DelegationStorage3.sol`,

which contains the data structure seen above, authorized users and institutions, and the hash and compare string methods.

4.4 Gas fees comparison

To evaluate the efficiency of the contracts in terms of fees, a comparison has been made on some basic functions which require fees to be invoked. The methods taken into account are:

- `addService`
- `delegate`
- `deleteDelegation` or `revoke`

In Table 1, the results are represented for each different contract. For the delegation and revocation functions, the code was tested on two times, to see how already existing delegations impact the execution.

Table 1: Gas fees comparison

Contract	add a service	1st delegation	2nd delegation	Revoking 1st delegation	Revoking 2nd delegation
Delega.sol	0.00017591	0.00047542	0.00044061	0.00027448	0.00025157
Delega2.sol	0.00012217	0.00060627	0.00061308	0.00028973	0.00028988
Delega3.sol	0.00012217	0.00047788	0.00023004	0.00011226	0.00011381

As can be seen, the costs have undergone different stages, rising with the second version and lowering with the third one. Adding a service is now less costly, but since the way they are managed remains unvaried between the second and the third version the cost does not decrease further. The first interesting observation can be made on the delegation cost. Even though the code in the second version of the contract seems to be more optimized and with fewer loops, the fee to be paid has increased. This is due to the amount of variables to be written anytime a delegation is performed. Indeed, the support data used to fast access and allow delegations added a non-trivial amount of write operations, making the price of the execution higher consequently. This problem has been solved with the third version, where the first delegation fee is similar to the first one, but the second delegation is much cheaper. For the revocation, the difference between the first two versions is rather small, with the second version always performing worse because of the amount of support variable to be modified or deleted. Again, with the third version a great improvement has been made, with a price which is two times cheaper than before.

The results of these observations highlight how the third version outperforms the previous in terms of cost efficiency, representing the best solution developed.

4.5 Testing

Another important upgrade that has been made from the first version of the contract is the implementation of some test cases for the Delega3.sol contract, in order to ensure its correctness, functionality, and security. 9 tests have been implemented in the file Delega.js and stored in the `/truffle/test` folder.

4.5.1 Test descriptions

- `Adding user:`

  Ensures that the Delega3 contract successfully adds a user. It calls the `addUser` function and checks if the user is authorized.

- `Adding an Institution:`

  Verifies the successful addition of an institution to the contract. It calls the `addInstitution` function and checks if the institution is authorized.

- `Delegating a Service:`

  Ensures that a user can delegate a service to an institution. It calls the `delegate` function and checks if the delegation is added successfully.

- `Checking Delegation Status Before Revoking:`

  Ensures the status of the delegation before revoking it. It calls the `checkDelegationUser` function and asserts that the delegation is present.

- `Revoking a Delegation:`

  Ensures that a delegation can be successfully revoked. It calls the `revoke` function and checks if the delegation is revoked.

- `Checking Delegation Status After Revoking:`

  Ensures the status of the delegation after revoking it. It calls the `checkDelegationUser` function and asserts that the delegation is not present.

- `Adding a Service to an Institution:`

  Ensures that new services can be added to an institution. It calls the `addService` function and checks if the service is added successfully.

- `Getting User Delegations:`

  Verifyates that the user can view his delegations. It calls the `userDelegations` function and ensures that there are no unexpected delegations for the user.

- `Getting Institution Delegations:`

  Ensures that an institution can view its delegations. It calls the `institutionDelegations` function and ensures that there are no unexpected delegations for the institution.

To run the tests, type `truffle test` in the terminal. This will cause the test suite to be executed and feedback on whether each test case was successful or not will be displayed. This is recommended to keep executing the tests regularly after any changes in the code in order to ensure the continued functionality of Delega3.sol.

5 Conclusion

Throughout the course, the development using the solidity language has been taken into consideration. Being Solidity designed to develop Ethereum's smart contracts, this project is designed to work on this public ledger. However, it is important to make some considerations about the various options and their advantages and disadvantages.

The characteristics of a public blockchain technology impact our application either positively or negatively. In the first place, writing on a public ledger means, as studied above, that some gas fees have to be paid. As a result, whenever a user tries to delegate someone or delete a delegation, they have to pay these fees in order to have their transaction processed, and the amount to be paid heavily depends on the congestion of the network when the contract call is sent. This represents the greatest drawback of such a solution because the users are forced to pay to use our platform, which is meant to be free to use for every delegator. There are many possible ways to approach this problem. The company can choose to take the burden of refunding all the payments made by the users (which is not really feasible) or can define under which parameters network usage fees are refunded or not. Similarly, the company can choose to ignore this cost, leaving the choice to each institution, which will consider whether to leave it to the users or not. Besides this, whenever a new user or institution subscribes to the Dapp, a fee is paid by the contract owner (the company) to invoke the `addUser` or `addInstitution` method. This could also be overcome by adjusting the costs of the platform.

However, Ethereum represents the standard for smart contract development and deployment, and it is also a permissionless technology that enforces the decentralization of governance and the distribution of computation, besides, of course, the immutability of the data. These aspects are the core concepts of why blockchain technology is being applied in many different fields and can bring added value to our service. Alongside these intrinsic properties of a public blockchain, comes the need to make data meaningful only through our platform. Indeed, data in the blockchain is public and everyone can read it. For this reason, an encoding and decoding system needed to be designed in the backend API so that only encrypted services are stored in the blockchain.

In conclusion, the contract is now complete with all its intended functionalities. However, since nowadays delegations are still free and require at most to print a document, adding so many costs to every operation can become a huge drawback for the adoption of our solution. A valid option to avoid all these overhead costs hence is to deploy the service in a private or permissioned blockchain maintained by the company. Among the possibilities, an interesting starting point could be a deep delve into technologies like Hyperledger Fabric or R3 Corda. These solutions could bring different benefits since code development would not need to be done taking into account any kind of fee, and the data structure could be designed more powerfully, opening the horizon to a broader range of functionalities. So, private or permission blockchains are probably the best option to implement this application.