

# Indisputable GDPR compliant signatures in ContractChain

18th August 2019

*To obtain the bachelor's degree*

*Authors:*

Martijn J.W. van den HOEK

Hendrik M. HOUWING

Frank C.J. VOLLEBREGT

*Coach:*

Dr. Zekeriya ERKIN

*Client:*

JORDI RÖER

LIZARD GLOBAL

*Project Coordinators:*

Ir. Otto VISSER

Dr. Huijuan WANG



## Abstract

Nowadays, entering into a contract with an overseas company still relies on postal services to send a printed contract, which is signed on paper. Lizard Global is developing an online platform for constructing, reviewing and signing digital contracts for one of their clients. In the original system, when a signee signed a contract, his personal information was used as a signature and stored in blockchain. However, this way of signing a contract does not enjoy the same degree of legal validity as a written signature. Moreover, the implications on privacy legislation, specifically the European Data Protection Regulation (GDPR) had not yet been taken into account by Lizard Global.

This project describes how agile development was used to construct a high quality software solution to the problem, thereby implementing firstly an advanced e-signature to make signing a contract legally binding and secondly functionality to store this signature in blockchain such that it is compliant with the GDPR legislation. This is done by only storing hashed values in the blockchain and adding a user panel. In this panel, signees are able to control their personal data. High quality is obtained by testing thoroughly (100 per cent branch coverage), using the static analysis tool ESLint and requesting, receiving and implementing feedback from the software improvement group.

# 1 Preface

This project is done as part of the Bachelor End Project (BEP) course, the final course of the Computer Science and Engineering (CSE) bachelor's degree at the Delft University of Technology.

Although we have worked on multiple large projects during our bachelor's degree, this is the first project where we are given the opportunity to work alongside people at an actual company. In our case, this is Lizard Global, a software development company based in Rotterdam and Kuala Lumpur. We worked together with the development team in their development centre in Malaysia. This also entailed adapting to their way of working. In our case the company, similar to how previous courses in our study programme taught us, employs an agile development methodology, in which large tasks are broken up into smaller chunks, such that they can be implemented during a single development sprint. The company uses sprints of two weeks, which we also adopted.

As becomes clear while reading this report, challenges were encountered along the way. The way we handled them is also described. We hope this report can help the reader to attain a better understanding of the project, our approach and our choices.

We would like to specifically thank everyone who helped us to complete this project. First and foremost, Jeremy Raes, CEO of Lizard Global, for giving us the opportunity to do a project at the company. Secondly we want to thank Jordi Röer, who conducted the role of Project Owner and was our main point of contact and feedback within Lizard Global, for his advice and support. We would also like to thank Dr. Zekeriya Erkin for coaching us from Delft, for bringing us in contact with Lizard Global, for providing feedback on our reports and pointing us in the right direction to make the project a success. Lastly, we would like to thank Khaldoun Alwan, Umer Saleem and Chathuri Senanayake, for managing the project and assisting us in the process of understanding the existing product and developing alongside it.

# Contents

<b>1</b>	<b>Preface</b>	<b>2</b>
<b>2</b>	<b>Introduction</b>	<b>5</b>
<b>3</b>	<b>Problem definition</b>	<b>5</b>
3.1	Problem description . . . . .	5
3.2	Constraints on the solution . . . . .	6
3.3	Success criteria . . . . .	6
<b>4</b>	<b>Background knowledge</b>	<b>7</b>
4.1	Asymmetric cryptography . . . . .	7
4.2	Hashing . . . . .	8
4.3	Blockchain technology . . . . .	8
4.4	Google Firebase . . . . .	9
<b>5</b>	<b>Problem analysis</b>	<b>10</b>
5.1	Requirements on digital signatures for legal indisputability . . . . .	10
5.2	Requirements for GDPR compliance . . . . .	11
5.2.1	GDPR principles . . . . .	11
5.2.2	GDPR guidelines . . . . .	11
5.3	The implications of GDPR and valid signatures on the current system . . . . .	12
5.4	Possible solutions . . . . .	13
5.4.1	Signing . . . . .	13
5.4.2	GDPR . . . . .	15
5.5	Proposed solution . . . . .	17
<b>6</b>	<b>Solution design</b>	<b>17</b>
6.1	Solution . . . . .	17
6.1.1	Signing the document . . . . .	18
6.1.2	Inviting signees . . . . .	18
6.2	Process of design . . . . .	19
6.2.1	Identifying system requirements . . . . .	20
<b>7</b>	<b>Process</b>	<b>20</b>
7.1	Development . . . . .	20
7.1.1	Sprints . . . . .	20
7.1.2	Git usage . . . . .	21
7.1.3	Google Firebase Deployment . . . . .	22
7.2	Quality assurance . . . . .	22
7.2.1	Testing . . . . .	22
7.2.2	Linting . . . . .	23
7.2.3	Continuous Integration . . . . .	23
7.2.4	Software Improvement Group . . . . .	24
7.3	Integrating with blockchain . . . . .	26
7.4	Working with existing code . . . . .	26
<b>8</b>	<b>Results</b>	<b>26</b>
8.1	Implemented system . . . . .	26
8.1.1	Signing a contract . . . . .	26
8.1.2	Updating information . . . . .	30
8.1.3	Deleting information . . . . .	31

8.1.4	Divergence from initial design . . . . .	32
8.1.5	Adapting existing code . . . . .	32
8.2	Testing and Eslint report . . . . .	34
8.3	Unimplemented requirements . . . . .	34
8.4	Personal growth . . . . .	34
<b>9</b>	<b>Ethical analysis</b>	<b>35</b>
9.1	Ethical issues concerning the means of signing . . . . .	35
<b>10</b>	<b>Conclusion</b>	<b>36</b>
<b>11</b>	<b>Discussion</b>	<b>36</b>
11.1	Personal assessment . . . . .	36
11.1.1	Evaluation F.C.J. Vollebregt . . . . .	36
11.1.2	Evaluation H.M. Houwing . . . . .	37
11.1.3	Evaluation M.J.W. van den HOEK . . . . .	38
11.2	Group assessment . . . . .	39
11.2.1	Team performance . . . . .	39
11.2.2	Implemented solution . . . . .	39
11.3	Recommendations . . . . .	39
11.3.1	Portability . . . . .	39
11.3.2	GDPR compliance . . . . .	40
11.3.3	NoSQL . . . . .	40
11.3.4	Data duplication . . . . .	40
<b>A</b>	<b>Database Structure Diagrams</b>	<b>43</b>
<b>B</b>	<b>Requirements in MoSCoW format</b>	<b>46</b>
<b>C</b>	<b>Component Diagrams</b>	<b>49</b>
<b>D</b>	<b>Feedback SIG</b>	<b>52</b>
D.1	Feedback on first upload . . . . .	52
D.2	Feedback on second upload . . . . .	53
<b>E</b>	<b>Eslint reports</b>	<b>54</b>
<b>F</b>	<b>Redundant requirements</b>	<b>56</b>
<b>G</b>	<b>Info sheet</b>	<b>57</b>
<b>H</b>	<b>Original project description</b>	<b>58</b>

## 2 Introduction

The Dutch company Lizard Global (formerly known as Lizard Apps) has been developing an application called "ContractChain" for one of their clients. This application aims to provide a method to digitally sign and store legally binding contracts in an immutable manner. This implies that once a contract is signed by a signee, no further changes can be made, as the state in which it was signed is retained indefinitely. A prototype of the product has been developed during a first development phase and Lizard Global is preparing a second phase.

Nowadays, most contracts are still signed on paper. Areas such as real estate and employment largely rely on this traditional way of solidifying an agreement. Paper contracts, however, have one significant downfall: When two companies located in different countries enter into a contract, sending a paper contract can easily take up multiple days, during which both parties are waiting for the contract to be delivered. Time is wasted, which limits advancement. By digitising the signing of a contract, the dependency on the postal services is taken out of the loop and therefore the amount of time wasted waiting is decreased. Lizard Global's client aims to provide an online platform where contracts can be constructed, reviewed and signed.

The signatures that are submitted through the application are stored in the blockchain, thereby ensuring that they are immutable. As Lizard's client intends to market the application to European customers, it needs to adhere to the European General Data Protection Regulation (GDPR). This regulation enforces rules governing which data can be collected and the storage and processing of personal information of citizens residing in the European Union.

The project revolves around the development of an extension to the existing code base, adding functionality to digitally sign documents such that the signatures are non-disputable on legal terms, while being compliant with the GDPR. Lizard Global must receive a piece of software with this functionality. This software must be of high quality, which is ensured by means of unit testing and static code style analysis. Moreover, agile development must be used to ensure constant communication and feedback between the developers, the project manager and the product owner.

## 3 Problem definition

The problem that needs to be solved is bipartite: It concerns how to digitally sign a document as well as how to store the resulting signature in a GDPR compliant manner. In this section, a formal definition and the constraints on the solution will be discussed. Additionally, the success criteria for the product will be specified.

### 3.1 Problem description

A prototype of ContractChain was developed during its first development phase. During this phase, the developers and designers at Lizard Global have built a web application to build contracts, invite signees, and let them sign from their own computer or mobile device. In this version, signing a contract implies pressing a green button, after which the signee's personal information is stored to indicate he signed the contract.

The basic implementation of the first development phase was neither required to be GDPR compliant nor to include legally indisputable signatures. The product is required to have both properties when it is finalised, so Lizard Global is looking for a solution that solves these two problems. In the following subsections, a detailed description of these problems is provided.

**Digitally signing a document** In the current system, the signing process is very basic and does not attempt to link the user to the document at all. How can a better signature be created when a user signs a contract, and how can it be assured that this signature is legally binding?

**Storing a signature** To achieve immutable storage of contracts, the application employs blockchain technology. Once the data is stored, its state is kept forever. How can one store the data such as names and signatures while respecting active privacy legislation, and more specifically, the European General Data Protection Regulation (GDPR)?

The motive for the client to achieve the indisputability of signatures is to strengthen the application in the core functionality. As the application's main purpose is for people and companies to digitise the processes of signing contracts, doing so without the signatures being non-disputable would mean that none of the signees of a contract can be sure that the document they signed is legally binding. Thus, signees would have no incentive to use ContractChain, and the client would end up with an unusable product.

Lizard Global's client has a financial incentive for the application to be GDPR compliant. As the application will be marketed to residents of the European Union, it needs to comply to the European privacy regulation. Offering a product that fails to comply with said regulations can result in significant financial penalties for the client.

### 3.2 Constraints on the solution

Lizard Global imposes constraints on the solution that can be implemented for ContractChain. These constraints have played an important role in the solution design and implementation process.

Firstly, the chosen solution must be implemented using the programming language that has been used to develop the initial version of ContractChain. This initial version has been built using Node.js [17] and the React library [22], meaning that the programming language used is JavaScript. The JavaScript language is enriched by JSX, a syntax extension of JavaScript used with React to more easily describe what the User Interface (UI) should look like.

Secondly, Lizard Global constrains the budget and time that is available for the solution to be implemented. This means that any chosen solution must be feasible within the planning of the project, both in terms of initial development and later maintenance by the developers of Lizard Global. This also means that the chosen solution must not lead to substantial additional expenses for Lizard Global or its client. For instance, this limitation prohibits the usage of licensed software or the requirement of additional hardware in the chosen solution.

Additionally, in line with the previous constraints the chosen solution must be within the technical capabilities of Lizard Global. Once the project has been delivered, its development team must be able to maintain, upgrade and extend the delivered solution.

Lastly, the implemented solution must be a web application, allowing users to access it from multiple devices. Lizard Global's client does not want its users to be restricted to a single device for using the application. This limitation implicitly means that the implemented solution cannot rely on information stored on a user's device.

### 3.3 Success criteria

The project can be considered a success only if the solution is a fitting one. Firstly, it must solve the problem for Lizard Global and its client. Additionally, it needs to fit the current

project, thereby not breaking any existing functionality. The performance and stability also have to be satisfactory for the client. Finally, the written code has to be of high quality and well documented and tested, such that it behaves as expected and the developers are able to continue coding without intensive research to the changes.

## 4 Background knowledge

In this section, general background knowledge on techniques discussed in later chapters will be provided. First, the concept of asymmetric cryptography will be explained. Next, generating a 'fingerprint' of a document by means of hashing will be discussed. After this discussion, background knowledge on blockchain technology as a means to store data in an immutable manner will be provided. Lastly, the features of Google Firebase which have been used to develop the project will be explained.

### 4.1 Asymmetric cryptography

The analogy of a modified padlock can be used to explain the concept of asymmetric cryptography. Suppose there exists a padlock that accepts two codes. One of those codes is made public, referred to as the *public* code. The other code is held private by the owner of the padlock, and it is referred to as the *private* code. The padlock together with its codes possesses a number of unique properties. Firstly, for each padlock there exists only one *pair* consisting of a *public* and *private* code. Entering any other code into the padlock will not have any effect. Secondly, both codes can be used to lock the padlock, but once one has been used to lock it, only the other can be used to unlock it. This means that if the padlock has been locked using the public code, it can only be unlocked using the private code and vice versa. Lastly, the public and private codes are *distinct* or *asymmetric*. This means that for no such padlock, it can be the case that the public and private code are the same. In conclusion, the padlock has a *key pair* of *asymmetric* codes to lock and unlock it. Now it is possible to do *cryptography* (locking and unlocking the padlock) using the *asymmetric key pair* consisting of the public and private code. This concept is known as *asymmetric cryptography*, in which case the public and private codes are commonly referred to as the *public key* and the *private key* respectively.

Asymmetric cryptography can be used to prevent third parties from reading transmitted information. Suppose Alice wants to send a message to Bob such that Charley, who works at the post office, is not able to read her message. Both Alice and Bob possess a number of the modified padlocks as described before.

In advance, they exchanged a padlock with each other: Bob gave Alice one of his padlocks and Alice gave him one of hers. When Alice wants to send a secret message to Bob, she puts the message into a container and secures this container using his padlock. To lock the padlock, she enters Bob's public code, which he shared with the public. When Charley intercepts the container, he is not able to open it as he does not possess the private code that opens the padlock. Bob however, upon receiving the container, can open the padlock using his padlock's private code, to which only he has access. This way, the asymmetric cryptography can be used to transfer messages without a third party being able to read the messages.

Another use of asymmetric cryptography is to establish the identity of a message's sender. Again, suppose Alice wants to send a message to Bob. In this scenario however, she does not care about whether Charley can read the message. Instead, when receiving a message that seems to be sent by Alice, Bob wants to be able to verify that Alice really is the sender of the message. In other words, Charley must not be able to send a message to Bob while pretending to be Alice such that Bob believes the counterfeit message was sent by Alice.

To achieve this, Alice again puts the message in a container and secures the container with one of her own padlocks. This time, she uses her own private code to lock the padlock. When she



sends the locked container, anybody can open it, as Alice's public code is known to everybody. Now, when Bob receives a locked container, he can easily establish whether or not Alice really is the sender. When he can open the container using Alice's public code, the padlock must have been locked using Alice's private code, as no other code besides Alice's public code is able to open the padlock when it has been locked with her private code. In the case that Charley tried to send Bob a counterfeit container pretending to be Alice, he must have locked the padlock using another code than Alice's private code, as only Alice has access to this code. As a consequence, Bob will not be able to open the padlock using Alice's public code, as no other code but the counterpart of the locking code can be used to unlock the padlock. Thus, when Bob can open a received container using Alice's public code, he can be sure that Alice really is the sender of the message in the container.

## 4.2 Hashing

Using the process of hashing, the digital equivalent of a fingerprint can be constructed for a piece of data such as a document. Suppose Alice has left fingerprints from her right index finger on multiple locations. Bob, when finding such a fingerprint, cannot know that the found fingerprint belongs to Alice. However, when he encounters Alice, he can take the fingerprint of her right index finger and obtain the same result each time. Taking fingerprints is a *one-way function*: it is easy for Bob to go from Alice to her fingerprints, but it is impossible for him to go from a given fingerprint back to Alice, unless he already took her fingerprint and remembered it. When Alice presents her thumb instead of her index finger, Bob will find another fingerprint than he would otherwise. Thus by changing a small aspect of the source, the resulting fingerprint can differ significantly.

The same process can be applied to electronic documents (and data in general): using a *hashing algorithm*, the *hash* (digital fingerprint) of a document can be established. This will result in the same hash each time this algorithm is applied to the same document. However, the hash of the document will change when the document changes. Thus, it is easy to go from a document to its hash. However, it is impossible to go from a given hash to the document it was created from.

## 4.3 Blockchain technology

Blockchain is an emerging technology that is used for storing data in an immutable manner, such that it cannot be modified after it has been stored. It is commonly known from so-called cryptocurrencies like Bitcoin, Ethereum and Ripple. In such applications, its unique property of immutable storage is used to keep a chronological list of all transactions that have occurred in the network.

Blockchain's immutable property is both its advantage and its disadvantage. On the one hand, everyone is able to verify the presence of certain data in a blockchain, because blockchains are generally public. On the other hand, one should be careful which data to store in a block chain for the same reason: Once stored, it can always be retrieved.

Blockchain technology works by chaining a newly added block to the full history of the block-chain. Each block in the chain contains at least two fields: a field containing the information that is added to the block chain and a field containing the hash of the previous block. When a new block is added, the hash of the previous block (containing the data of that block and the hash of the second last block) is taken and added to the newly added block. A conceptual representation of this process of chaining blocks has been depicted in Figure 1.

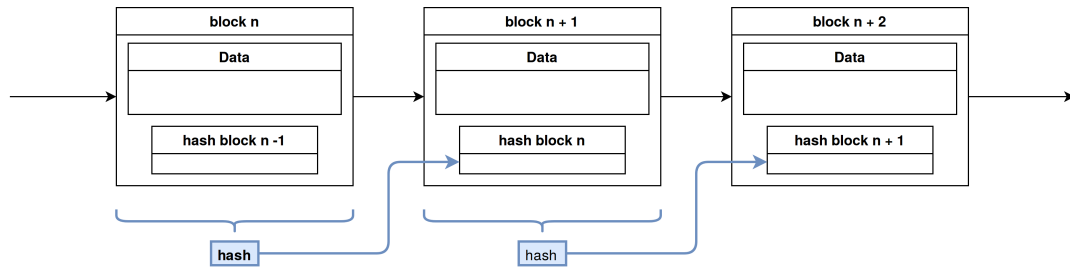


Figure 1: Conceptual representation of blockchain technology

Blockchain’s immutable property originates from the chaining of blocks and the effort involved in adding a block to the chain.

By including the hash of the previous block in every newly added block, it becomes impossible to modify an already added block without modifying the block that comes after it. Suppose a person wants to modify block  $m$ . Modifying block  $m$  would change its hash, causing a mismatch between the hash of block  $m$  and the hash that is stored in block  $m + 1$ . Updating block  $m + 1$  would change its hash as well, so modifying a single block in the chain requires a ripple of changes that modify every block after the single one that was modified originally. Would it be possible to add a new block to the blockchain without significant effort, modifying a block and all subsequent blocks would be trivial. To prevent such events, blockchains require significant effort to add a new block to the chain.

#### 4.4 Google Firebase

Google Firebase is an online platform that is used throughout the project for storage and hosting. This subsection briefly describes the Firebase functionality that is used during the project.

**Authentication** Firebase authentication contains built-in functionality to easily authenticate users. This includes things such as sign in, sign up, email verification and password reset options.

**Realtime Database** The Realtime Database provides cloud-hosted NoSQL storage and retrieval of data in JSON (JavaScript Object Notation) format. It allows for flexible storage where no strict rules apply to the format of the data that is stored, as is conventional for NoSQL databases.

**Cloud storage** The cloud storage provides a storage solution for binary files, such as images, videos and documents. When stored, the Firebase API returns a URL that can be used to retrieve the data. Because data can also be stored in nested folders, it is easy to keep data organised. Data is stored in so-called ‘storage buckets’.

**Hosting** Firebase hosting allows easy deployment of the application such that it can be accessed from the web. All client-side pages are loaded and served from the Firebase hosting.

**Cloud functions** The cloud functions are server-side functions that can be triggered on Firebase events: They can listen to certain activities in the Realtime Firebase, or when configured, they can be called directly to trigger an external API call to, for instance, send an email.

## 5 Problem analysis

This section will analyse the problem to be solved. This will be done by carrying out analysis on the two sub-problems of the project as described in section 3. The first part of the analysis will research what is required of a digital signature to make it legally indisputable. The second part will focus on what it entails to be complaint to the GDPR. After both analyses have been completed, the implications of both requirements for the current implementation of Contract-Chain will be examined. Then, multiple potential solutions to the problem will be proposed and evaluated on their strong and weak points. Finally, the chosen solution will be discussed in more detail.

### 5.1 Requirements on digital signatures for legal indisputability

As far as traditional paper contracts are concerned, a signature on paper is sufficient to make a document legally binding. For a digital signature to have the same level of legal validity, it needs to meet additional requirements. In section 4, article 26 and 28 of the regulation of the European Union [20], a distinction is made between two different types of digital signatures:

**Standard electronic signature** This is essentially a digital version of a signature on paper. It is the most basic version used to identify an individual. It can for instance be a photograph or scan of a handwritten signature, a signature directly written on a computer or mobile device, or a typed out name.

Although such a standard signature shall not be denied legal effect or admissibility as evidence in legal proceedings (as described in Section 4, article 25 (1) of Parliament and European Union [20]), this does not mean that it is considered as legally valid as a signature written on paper [25].

Key to classifying a digital signature solution as trustworthy is that it is able to register who signed what document and when the document was signed [16]. In the case of standard signatures, the solution answers who signed the document, but not what was signed. An adversary could obtain the signature of a client and paste this under a counterfeit document, without this being detectable.

**Advanced electronic signature** The advanced electronic signature, on the other hand, requires the fingerprint of the document to be signed using a unique key, which assures the authenticity of the signed document and the identity of the undersigned person, making this type of electronic signature more legally binding. Depending on the origin of this key, a different level of validity may be achieved: When the key originates from a qualified trust service provider (TSP) in accordance with the requirements described in Annex I, *Requirements for qualified certificates for electronic signatures* of the EU regulation [20], the advanced electronic signature is considered a *qualified* electronic signature.

Such a qualified electronic signature has the same legal validity as a handwritten signature, as described in Section 4, article 25 (2) in *Legal effects of electronic signatures* [20].

## 5.2 Requirements for GDPR compliance

In this section, the GDPR legislation is analysed. The rules and rights relevant to the project will be discussed.

### 5.2.1 GDPR principles

GDPR describes six principles related to processing of personal data in article 5(1) [19]:

1. Personal data has to be processed lawfully, fairly and transparent.
2. There must be a specific purpose for processing personal data and this purpose must be indicated.
3. Only data that is necessary for an indicated purpose can be collected.
4. Personal data must be accurate and should be kept up-to-date.
5. Personal data is not allowed to be stored longer than necessary. The time data is stored should be indicated and should be the shortest time possible.
6. Personal data shall be processed and stored in a secure manner.

### 5.2.2 GDPR guidelines

In this section, guidelines given by the EU for businesses and organisations will be discussed to find out which steps need to be taken in order for the storage of a user's signature to be GDPR compliant. Only the guidelines relevant for this project will be discussed and the guidelines that are not applicable to the implementation of the project will be omitted. Examples of such inapplicable guidelines are those that describe the exact clauses that need to be incorporated into the terms and conditions and the usage of automated decision-making. As it is not in the scope of this project to compose the terms and conditions for the software, the first example does not have to be taken into account. However, it will be communicated to Lizard Global that such terms and conditions are to be included, and the client will be briefed about the information that the implemented solution retrieves and processes. Signing of a contract does not require automated decision-making, therefore this regulation is not needed for the project either.

**When consent is withdrawn** When consent is withdrawn, all personal information must be deleted. The main exception to this rule is information that is still required for legal reasons[5]. An option for the user to revoke their consent for his data being processed must be implemented

**Data protection by default and by design** Personal data must be protected by design of a system [4]. This means that the system must have safeguards in place to protect the data of the users. This could include the use of pseudonyms to cover the real identity of a user, or the use of encryption to encrypt the personal data. Protection by default means that all the default values in the system are set to protect the privacy of the user. In the scenario of a social media platform, uploaded pictures or publicised posts should by default only be visible to the friends of the user, unless the user himself decides to make them public. For this project this implies using a recent and secure hashing algorithm to minimise the risk of any data being compromised.

**Right of access** When somebody requests their data [6] and has previously signed one or more contracts, the application must return a copy of the signatures as well as a copy of all his signed contracts for free, but a reasonable fee can be charged for further copies.

**Right to be forgotten** Data has to be deleted on request, except when there is a legal obligation to keep the data [3]. This implies that the information of a user must be deleted, but that the signature of the user must be kept, as it is needed to show that the user signed a contract.

**Right of rectification** A user must be able to rectify his personal data, as is described by article 16 of Parliament and European Union [19]. This means that a user needs to be able to update his account information. However, the legally binding contract can only be updated when all involved parties consent to the update.

**Right to object to processing of personal data** A user can object to the processing of his data. When this happens, the company may no longer process the personal data of the user, unless it can demonstrate that it needs to process the data for reasons that override the rights of the users or if this processing is needed for legal reasons [9] [11].

**Right to restrict processing** A user has the right to request the restriction of processing of his personal data. This means that the data is not deleted, but that it may no longer be processed without giving permission. This right to restrict can be exercised when a decision for right to objection is pending [12] and article 18 of Parliament and European Union [19]. For the ContractChain, this would imply that the user's account remains intact, but that he/she can no longer sign contracts without explicitly giving consent.

**Right of portability** Users have the right to request the personal data that has been collected on them. When the data is returned to them, it needs to be structured in a machine readable format. Moreover, the company from which a user requests data needs to make sure that the data that is given back is easily portable to other platforms [10].

### 5.3 The implications of GDPR and valid signatures on the current system

This section will evaluate the current system against the previously established GDPR and the regulations regarding the legal validity of a digital signature. The concrete issues which need to be resolved are thereby derived.

The current method to sign a contract is to have the signee press a button. When this happens, the back-end stores a block in the blockchain that contains the signee's email, full name, phone number, a time stamp and the identifier of the block that contains the signed contract. This can be considered as a standard electronic signature (subsection 5.1): While it does store the signature of the user together with the document reference, this signature can easily be extracted and placed with the reference to another document. As described before, such a signature's legal validity can be challenged with relative ease compared to the other types of signatures, as they can be used by an adversary without this being detectable. A more elaborate version of the standard electronic signature, or an advanced electronic signature would greatly ease the effort required to prove the validity of the signature when a contract is disputed.

As is noted in subsection 5.2.2 in *right to access*, a user should always be able to retrieve the personal data which is stored. In the current system this is not possible, because the signee is not obligated to have an account. The data that is stored when a contract is signed cannot be requested afterwards. As a result, the system is not GDPR compliant. Note that this also

implies that the user is unable to retrieve his data to feed it into another service, and that therefore the system is in violation of the *right of portability* as well. As a user is unable to retrieve his data from the system, he is neither capable of verifying the information that is currently stored, nor is he able to correct any data that is incorrect or no longer up-to-date. This means that the current implementation of the ContractChain also violates the *right of rectification* as laid out in the GDPR.

Furthermore, the system in its current state is in violation of the *right to withdraw consent*. Although it would be possible to ask a signee for consent for the processing of his data when he goes to sign the contract, it is not possible for him to easily revoke this consent, as no account is created for him in the system.

## 5.4 Possible solutions

In this section, possible solutions will be discussed. The solutions will address the problems as described in section 3 and the found issues in the current system of ContractChain as described in subsection 5.3. The found disadvantages of each solution will also be named. These solutions have been discussed with the Project Manager and Lead Developer to determine their feasibility. The next section (subsection 5.5) describes the chosen solution in more detail.

### 5.4.1 Signing

**Standard e-signature** A signee can draw a signature on his mobile phone or in a web browser. The signature is sent to the application. Together with contract and personal data (email, phone number, time-stamp), the signature is added to a block and added to the ContractChain. When a signee wants to retrieve his data, the signature is added to the contract before it is sent back to him. The digital version of a handwritten signature can help to show that the intended signee was the one who signed the contract. The personal data that is part of the signature could be extended to include the IP address from which a contract was signed, to further strengthen the identity of the signee.

**Disadvantage** As explained in 5.1, standard e-signatures do not enjoy the same legal power as a handwritten signature on paper. This way of implementing the signatures would imply that the produced signatures are not sufficiently strong to use as evidence in a legal case.

**Advanced e-signature** In order to implement the functionality of signing contracts in ContractChain, an advanced e-signature can be used. The following steps are needed to integrate this signature in the application:

1. Use the RSA [23] algorithm to generate a public and private key-pair for the signee each time he is requested to sign.
2. Send the public key to the server over a secured connection.
3. Hash the document using the SHA-3 hashing algorithm [8].
4. Use the private key to encrypt the document hash.
5. Send the encrypted hash and signature to the server to be stored in the blockchain.

Now the server can verify authenticity with the public key of the signee. The server can also check if the document is altered by checking the hash. The encrypted hash and a reference to the blockchain address of the contract is added to a block. The block is added to ContractChain. In the GDPR it is specified that a person is able to request all data stored about him. When a signee wants to retrieve his data, the signature is added to the contract before it is sent to them.

**Disadvantage** As explained in 5.1, advanced e-signatures do not enjoy the same legal power as an handwritten signature in EU legislation, although in court it is preferred above the standard e-signature.

In order to link multiple contracts to the same signee, two approaches can be implemented. Either the private key has to be encrypted and stored on the ContractChain server, opening the door to breaches. Another solution is to generate a new key-pair for each contract that needs to be signed.

**Qualified advanced e-signature** In order to implement the functionality of signing contracts in ContractChain, a qualified e-signature can be used. The following steps are needed to incorporate this signature in the application:

1. Use the RSA [23] algorithm to generate a public and private key-pair for the signee each time he is requested to sign.
2. Get a certificate from a trusted service provider (TSP).
3. Send the public key to the server over a secured connection.
4. Hash the document using the SHA-3 hashing algorithm [8].
5. Use the private key to encrypt the document hash.
6. Send the encrypted hash and signature to the server to be stored in the blockchain.

As is the case with advanced e-signature, the server can now verify authenticity with the public key of the signee. What makes the qualified signature more potent than the advanced one is that in the qualified version there is a third party (the TSP) involved that vouches for the accuracy of the identity of the signee. The server can also check if the document is altered by checking the hash. The encrypted hash and a reference to the blockchain address of the contract is added to a block. The block is added to ContractChain. In the GDPR it is specified that a person is able to request all data stored about him. When a signee wants to retrieve his data, the signature is added to the contract before it is sent to the user.

The private and public key need to be stored on a server together with the personal data of the signee. When the same signee is again asked to sign a contract, the same private key can be used to sign the contract.

**Disadvantage** Using qualified advanced e-signatures would require each signee to obtain a certificate from a qualified trust authority. As this takes the signee time and costs money, the overall user experience suffers significantly. This is especially the case when a user does not have a ContractChain account and needs to sign only one contract.

Furthermore, the user needs to be able to take his private key (connected to the certificate) everywhere, unless he obtains a separate key for each device. This means that the private key will have to be stored on the server in an encrypted form. Storing private keys of users on a server is generally seen as a bad practice.

**Advanced/qualified signature from a mobile application** As described before, both advanced and qualified signatures suffer from being deployed in a website, as this makes storing the private key of the certificate in a secure manner difficult. The advanced signature solves this by generating a new certificate for each contract, although this is not a very clean solution. By developing an application for smart phones that can be used to sign contracts, there would be a place to store a user's private key, eliminating this problem.

**Disadvantages** By requiring the mobile application to be used when signing a contract, a signee is hindered in his process of signing. This is not practical for users that need to sign only one document and are not yet part of the ContractChain environment. Besides this drawback, developing a mobile application would increase the complexity of the system.

**Other examined solutions** All solutions that have been proposed above either use no encryption (such as the standard signature), or use public key encryption (both advanced signatures and qualified signatures). Other means of achieving a digital signature have been looked into and are described below.

E-signature solutions that do not use public key encryption but rely on other means of authentication have been reviewed by Zefferer and Teufl [26]. As can be seen from this review, solutions that do not use public key encryption as their main way of authentication either force the user to be in possession of an enhanced SIM card or require the server to be equipped with a hardware security module (HSM). Implementing this solution would mean that users would be forced to obtain an enhanced SIM card, or that the servers that run ContractChain need to be equipped with an HSM. This would require a significant investment in the underlying infrastructure.

Another way of achieving digital signatures is by using biometric data to generate the key pairs, thereby eliminating the need to store the private key on the server when the key needs to be reused across devices (as is the case with qualified signatures). This approach has been described by Feng and Wah [13], but was joined by multiple drawbacks, preventing this approach from being a viable solution for the ContractChain application. The proposed system requires a user to draw his signature ten times when enrolling in the system. To be able to send an unknown user a request to sign a contract and have him draw his signature ten times would impose too much of a burden and would hurt the overall user experience.

Moreover, the performance of such a system is not good enough for commercial usage. There exists a trade-off between the false negative rate and the false positive. For instance: when a low false positive rate of 1.2% is opted for, this system has a 28% false negative rate, which means that in almost a third of all cases, when a valid signature is submitted, it is rejected, which is undesirable. When both rates are equal, they are both at 8% [13]. The relatively poor accuracy of this system is a second reason why such an approach is not suitable for ContractChain.

## 5.4.2 GDPR

**Signee Account** In order to comply with GDPR, the signee has to be able to have control over his data that is stored on the server. To allow this, the signee could have an account where he can control his data. When the document owner requests a signee for the first time, an account should be automatically made. The signee will be provided with a password which he can use to log in. When a document owner wants to request the same signee again, the account of the signee should be visible. Via this account the document owner should request the signee. In this way the personal data of the signee and the documents he has signed keep linked. In this account, the signee will be able to update and retrieve his information, which is not possible in the current implementation.

**Terms of service relating to GDPR** To comply with GDPR, certain aspects of the ContractChain need to be adopted into the terms of service (ToS). Those areas of special interest will be elaborated on next.

The user has the right to request deletion of his personal data, as enforced by the right to be forgotten. However, the contracts that the user signed also contain the personal data of the



user. As there is a legal justification (article 17, 3.(e) [19]) to maintain at least the contracts that are not expired, the ContractChain is not obligated to remove the personal data of the user from those contracts. This restriction of the rights of the user should be explicitly stated in the ToS.

Concerning the right to rectification, the user's rights should also be explicitly restricted by the ToS. As the user has the right to rectify the personal information that the system stores about him, he should be able to modify this data. However, it should not be possible for a user to modify an existing contract without explicit consent of all other signees of the respective contract as this would not be legal. This means that the ToS should state explicitly that the user has the right to modify his personal data that is not directly linked to the signature of a document. It should also state that modifying the personal data contained in a document is only possible when all other signees of the contract agree to the change.

The ContractChain system processes the personal information of the user in order to be able to sign contracts. According to the right to restrict processing, the user should be able to request the system to stop processing his personal data. It should be included in the ToS that when the user invokes this right, he can no longer sign contracts, as this would require the system to process his data. It should also be stated that when this right has been invoked, the system can still process contracts that have been signed before the request to restrict processing was received. The already signed contracts can still be processed as there is a legal need to maintain and process (article 17, 3.(e) [19]) the signature of the signees of a contract.

**Right to be forgotten** Once information is stored in the blockchain, one could argue that the right to be forgotten is violated. This is because information in the blockchain is stored indefinitely. As a consequence, once the information is stored in the blockchain it can never be deleted.

In the case of ContractChain, the second part of this premise holds true. That is, all information stored in the blockchain can never be deleted. However, when using advanced electronic signatures the GDPR is not violated, due to the nature of the type of information that is stored. The ContractChain blockchain would then only store two types of information: The hash of the contracts to be signed and the signatures of the signees who signed it.

For the hashes it can be argued that as the data is anonymised, it does not constitute personally identifiable data as described in recital 26 of GDPR [19].

While the contract is still valid, there is a legal obligation for the signatures of the signees to be stored in an immutable manner, such as in the blockchain.

Similarly, when the contract is expired, it is the purpose of the application to store the contract immutably, as this can still aid to provide proof that this contract was signed by a particular signee. The blockchain remains an excellent way to guarantee immutability of storage.

Storing the signature in a mutable manner would mean that the signature becomes weaker evidence in a court case, as it could have been manipulated.

## 5.5 Proposed solution

The following solution was proposed to the Project Owner: The creation of an account for all signees should be obligatory, the terms of service (ToS) should be extended and the advanced electronic signature based on the OpenPGP protocol should be implemented.

Creating an account for every user who has his information registered with the system allows him to access, delete or rectify his personal data. Granting every user an account allows the system to respect the GDPR regulations within the bounds described in section 5.4.2. The ToS should be extended to cover the cases where GDPR does not have effect due to legal reasons, such as requesting the deletion of an active contract.

The advanced electronic signatures can be implemented as follows. For each contract that the user needs to sign, he generates a new public/private key-pair and encrypts the hash of the contract with his private key to create a digital signature. He then sends the generated certificate (consisting of the public key) together with the signature back to the server.

The certificates can be generated using the OpenPGP.js [18] library that implements the PGP protocol, which uses asymmetric cryptography, in JavaScript. This library allows to manage PGP keys and to sign and validate signatures. The project is supported by the Horizon 2020 Framework Program of the European Union [7]. The library has been subject to an independent security audit [14]. At least the three critical vulnerabilities that were found during the audit were fixed<sup>1 2 3</sup>. The authors of this document have not checked if the other reported vulnerabilities were fixed as well.

The SHA-3 hashing algorithm can be used to hash the document. This algorithm is the newest standard for hashing as determined by the American National Institute of Standards and Technology (NIST) [8].

The certificates can be generated using existing libraries that have been reviewed by the cyber-security community. The usage of advanced electronic signatures on the bases of OpenPGP has multiple advantages over the current implementation of signatures. First of all, it allows to bind an unique signature to the document in such a way that any change to the signed document is detectable, as the hash of the document will be different when the document changed. Another advantage is that this way of implementation opens the door to even more potent solutions to the legality issues, might the need for this arise in the future. By making each user obtain his own certificate from a qualified trust authority, his identity can be proven even stronger. This however implies that the user always needs to have his private key available on all his devices. There does not exist a clean solution to store the private key locally in combination with a web-application.

## 6 Solution design

During the project, the two main problems indicated in subsection 3.1 needed to be solved. At the start of the project, a solution was designed and approved by the client. The first section explains the designed solution. The second section discusses the process of designing this solution.

### 6.1 Solution

In order to add a legally binding signature and store this in a GDPR compliant manner, changes to the system were needed. In this section these changes will be described and the made design choices will be justified.

---

<sup>1</sup><https://github.com/openpgpjs/openpgpjs/commit/329c92bc7375b8e1602637104915ae7596fcd16a>

<sup>2</sup><https://github.com/openpgpjs/openpgpjs/commit/93ca8b62fe04f6942966acae43efdd8e76df4b78>

<sup>3</sup><https://github.com/openpgpjs/openpgpjs/commit/e1fcc51d0eeacead3d5e2f9ae8cd0d448755fe02>

### 6.1.1 Signing the document

The following design was made to solve the problem concerning the immutable storing of a legally binding signature. To ensure the same rights as a written signature, the e-signature needs to make use of asymmetric key encryption. The qualified e-signature was not possible, as it introduced the need for a third party and extra financial costs to acquire a qualified certificate. Lizard Global indicated that this would be an issue. Instead, the advanced electronic signature was used in the initial design.

In order to allow a signee to sign a document, the signee generates a SHA-3 hash of the document. SHA-3 is used as it offers better security against some known attacks compared to the SHA-2 hashing family [8, Chapter A], because SHA-3 uses a sponge construction [1] to hash its values. This is what distinguishes SHA-3 from the SHA-2 family, as SHA-2 uses a Merkle–Damgård construction [21]. Although SHA-2 is not yet broken, there are advancements in constructing attacks against it [24] [15]. Therefore, the design uses SHA-3 to ensure that the personal information in the hashed contract, which is stored in the blockchain, can not be deanonymised. The signee then signs the SHA-3 hash with his private key obtained from the OpenPGP.js library. The public key is sent to the database where it is used to verify the signature. Using the user’s public key, the server decrypts the signature to obtain the document hash and verifies authenticity of the document hash and the user signing it. The signature can easily be verified for document authenticity as changes to a contract result in a different document hash.

This solves the issue of legally binding an e-signature to a contract, because by the properties of the asymmetric encryption used by OpenPGP only the person who is in possession of the private key is able to construct the signature.

### 6.1.2 Inviting signees

The original system did not comply with GDPR when considering the infrastructure around the signees. In the original system the signees were not able to request, update or delete their personal data, which are violations of GDPR (5.2.2). The design solves this issue by enforcing a personal account for signees. The process of doing this is further explained below.

When a signee accepts an invitation for the first time, he is directed to the sign-up page. After signing up, a new email with an invitation link to sign the contract is sent. Using this link, the signee is directed to signee page where the contract can be signed or declined. Figure 2 is a visual representation of the described process.

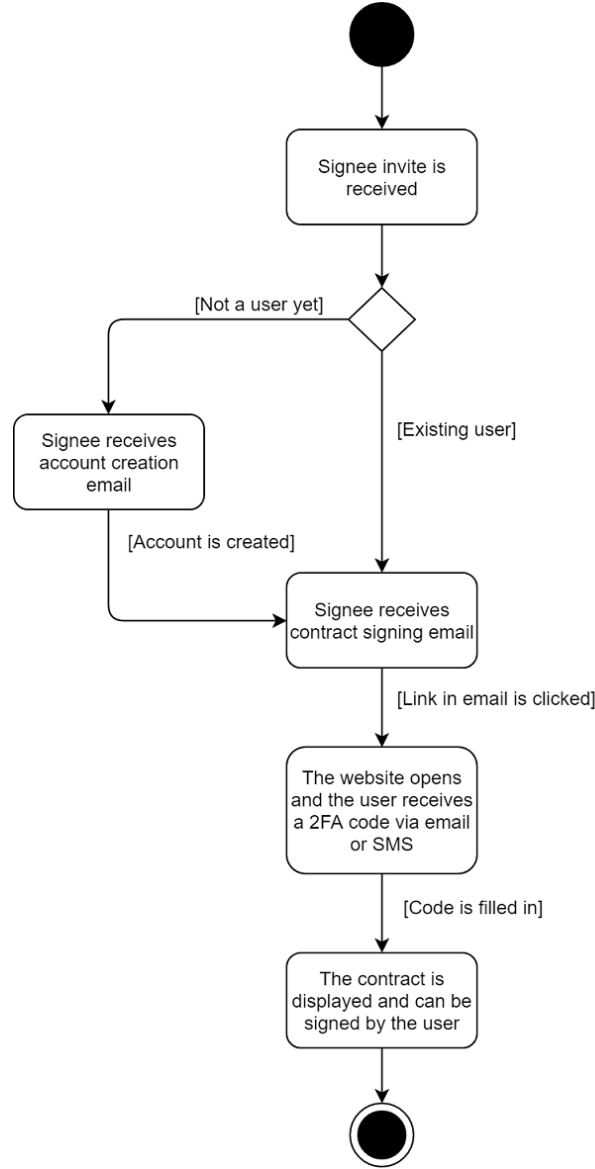


Figure 2: Activity diagram of the signee process

After signing up, the signee is registered as a signee in the database. With the created account, the signee is able to request, update and delete his data.

In this way, inviting a signee to sign a contract is done in a GDPR compliant manner, as the signee is now able to access his personal data via his signee account. Enforcing the signee to have an account requires changes to the original NoSQL database.

Figure 17 in Appendix A shows the proposed changes that were needed in order for the signee to have an account. In this design, there are two types of users: Employees and Signees. A user can be neither, either or both. Employees and signees are fields of the User table, referencing the Employees and Signees database tables respectively. The signatures are stored in the signees table. The signee ID is used to link contracts and invitations to signees. The invites table is used for both company and signee invites.

## 6.2 Process of design

During the design phase of the product, the freedom in design was mainly limited by the preexisting code base. The solution needed to fit the project, and not break the already implemented features. Moreover, there was no documentation or planning available and thus there was

no direct knowledge about preexisting functions and classes. As a consequence, constructing implementation oriented diagrams such as class diagrams, was deemed unfeasible.

### 6.2.1 Identifying system requirements

In order to find a fitting solution, first the required changes and additions needed to be identified to make the system legally binding and GDPR compliant. This was done by analysing the problem (chapter 5) and translating these enhancements to the current system. For instance, because of the GDPR, the system is obligated to implement an infrastructure that enables users to update their personal data. The translation to the system is the need to have a page where the user is able to change his/her data. These identified requirements were listed in a requirements document using the MoSCoW method. These requirements can be found in Appendix B. The changes were proposed to Lizard Global, who finalised the prioritisation of said changes.

## 7 Process

The designed solution was implemented using agile development. In this section, a detailed description of the development process is given. Additionally, the means to assure high quality of code will be discussed, an explanation of the integration with the blockchain will be given and the main challenge we faced while working with the existing code base is briefly highlighted.

### 7.1 Development

In this subsection, first the structure of the development sprints will be explained. Secondly, some changes in the solution design are highlighted, after which the usage of Git within the project is described. Lastly, the usage of Google Firebase is briefly explained.

#### 7.1.1 Sprints

The sprints were structured as depicted in Figure 3. During the sprint, there were two kinds of meetings: Sprint meetings with only the development team and client meetings attended by the development team, the project manager (PM) and the lead developer of ContractChain. This section will describe these meetings in terms of structure and focus points. Next, a global description of the development during the sprint is given.

**Sprint meetings** At the end of each sprint, a sprint review was held to review the progression of the project and which issues needed to be transferred to the new sprint. The team also determined which functionality from the requirements document needed to be completed the coming two weeks. From these requirements, issues were derived and added to the milestone belonging to the coming sprint. The issues contained a description of the issue, labels concerning the type of issue and the requirements for completing the issue.

**Client meetings** At the start of each week, a meeting was organised with the client, in which the planned changes for the upcoming week were discussed. Together with the planning, potential issues plus their possible solutions were addressed. A discussion with the lead developer was held to determine the optimal solutions. The meetings were also used to discuss the progress and to showcase newly added functionality.

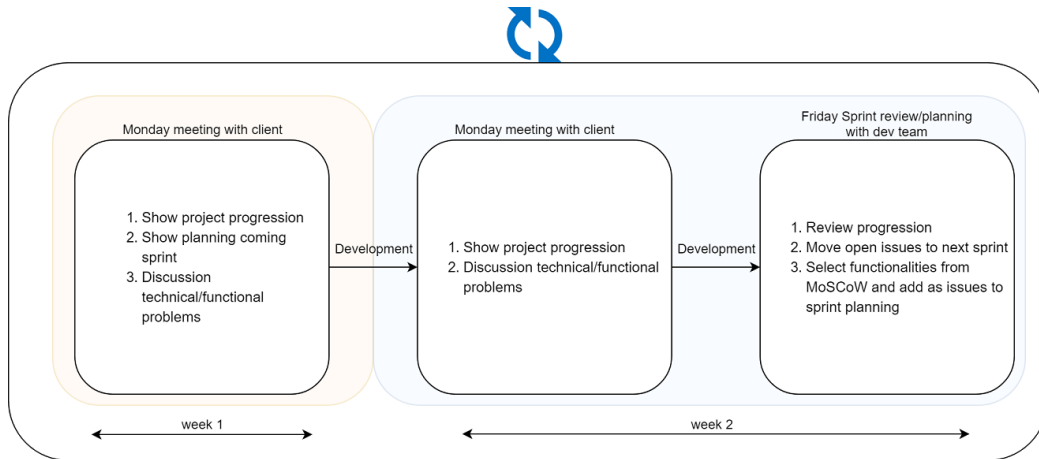


Figure 3: Sprint overview

**Development** During development, a sprint board was used to keep track of progress. The board consisted of three tables; firstly open and unassigned, secondly open and assigned and thirdly closed. Before starting the development, each team member determined which issue they would work on first. Whenever an issue was completed, a new issue was chosen from open and unassigned and moved to the open and assigned board. The developer was linked in the assigned issue. If a bug was found during implementing a feature, this bug was added as an issue, and the *BUG* label was assigned.

During the sprints, often decisions needed to be reevaluated because the team found out that the solution in mind did not fit in the existing code. A discussion was held in order to find a new solution that did fit.

There are two explanations why the initial design did not always fit the existing code. There was a lack of overview of the inner workings and lack of documentation of the original project during the design phase. This led to the inability of designing the product in detail beforehand. Also, since the team was relatively unfamiliar with NoSQL and JavaScript, there was little knowledge about their exact capabilities and limitations.

It was also sometimes required to rewrite pieces of existing code. Before making these changes, the plan needed to be communicated with the lead developer or PM. Skype was used as a main communication platform to have quick discussions about the implementation. The main communication within the development team was oral, as the team worked side by side in the same work-space. This made making decisions about, for instance, changing the initial design both goal-orientated and efficient.

### 7.1.2 Git usage

Lizard Global's GitLab version control system was used to keep track of changes in the source code of the application and to incrementally update the product using merge requests. At the beginning of the project, a separate branch was created from the master branch that was meant to serve as the master branch of this project. Thus, there was a central master and a project master branch. For each new issue, a new branch was created from the project master branch. Then, changes were implemented in this new branch. The branch was kept locally until the feature was done and ready to be merged into the project master. Then, the feature branch was rebased on the project master branch if this branch was updated while the new feature was being developed. After the rebase was completed, the feature branch was pushed and a merge request was created. At least one other team member had to review the merge request. When the merge request was approved by the involved reviewer(s), the feature branch was merged into the project master branch and subsequently deleted.

### 7.1.3 Google Firebase Deployment

Lizard Global uses Google Firebase to host and deploy their applications. For the project, this meant that the database, file storage, cloud functions and hosting were all available in the cloud. Cloud-based database and file storage ensured that those two services were always available, and no desynchronisation could appear between different clients or hosted instances.

Whenever a change was made to the cloud functions, they were redeployed from the development machine to the Google cloud servers. Once deployed they could be called from the client API by all clients. Logs for the purpose of debugging were automatically kept in the online Firebase console.

Firebase also provides functionality to host the web page. This is for instance used by Lizard Global to more easily showcase the application to the client without the need to deploy and connect to a self-hosted web server. During the project, locally hosting the web page was more sensible: Debugging is much faster because the application updates within seconds when hosting it locally. Moreover, no external access was ever required.

## 7.2 Quality assurance

To assure the code quality of the developed application, tools were adopted to test the code, enforce style consistency between source files and to assure that the application would run without needing any non-included dependencies. Furthermore, created code was uploaded to the Software Improvement Group, a third party that helped to analyse the code, identify potential quality issues and provide feedback on those. This section will explain each of the aforementioned techniques and how they were used.

### 7.2.1 Testing

For each source file that was created, testing was required with a minimal branch coverage of 80%. For each feature that was implemented, it was required that this feature was tested by isolating the components of the features and then testing each component individually (unit tests). If a file that was created had multiple paths that the software could take, these paths needed to be tested. A real life example of these paths would be the sentence "If there are less than five eggs left in the fridge, go to the store and buy eggs": there are two path to be taken in this sentence, either there are more than five eggs left and no action is required or it is required to buy new eggs. To call a file tested, at least 80% of all paths in the file had to be taken during testing.

However, changes in functions that were already written by developers at Lizard Global were not tested. The reason for this is that in the majority of cases the functions were not well isolated, which made testing without restructuring very difficult. The intention was to isolate the newly written code of the project as much as possible from the existing code in order to be able to test the added functionality. However, this was in some cases unfeasible. If the functions were restructured, the code from Lizard Global was also tested. This was done to be sure that no original functionality broke after the refactor.

For testing the front-end code, the Enzyme<sup>4</sup> utility was used. This utility facilitates functionality to render the visible part of the website and make assertions about what it looks like. For example, the user panel was rendered and it was asserted that it included the three correct buttons, one to download personal information, one to request the application to stop processing, one to change the personal information of the user.

---

<sup>4</sup><https://airbnb.io/enzyme/>

To test the back-end code, the Jest <sup>5</sup> framework was used. This framework can be used to run a function and make assertions about the outcome of the function, given a predefined input. In case of the eggs example, the function would be called with the argument "there are three eggs left" and it would be asserted that the function would return that new eggs were to be bought.

Isolation was achieved by mocking dependencies using the Sinon <sup>6</sup> framework. When some function *A* needs to be tested and it calls some other function *B* mid-way, it is not desirable to make a call to *B* each time while testing. Instead, Sinon is used to act like it is function *B* and return a predetermined result. This allows for testing all behaviour of function *A* isolated from function *B*. A practical example of this is a function that calls the blockchain to add a signature. The intention of the test is not to verify that the blockchain works as expected, only to verify that the function is using it correctly. To do so, the function that adds the signature to the blockchain can be mocked, meaning that it is replaced by a 'fake' function. In doing so, it is possible to make the blockchain function behave as desired and to verify that the function using it behaves as expected. For example, it is possible to make the mocked blockchain fail on purpose and then verify that the function under test handles this error correctly.

Besides the usage of automated unit tests, manual user testing was deployed to guarantee the integration with the existing product. ContractChain did not contain any tests before the project started, and validation was achieved by user testing, where the PM would act as a user to verify that a new feature was working. As the written unit tests only focused on testing the code that was written during the project, there was no automated way to validate integration of the project with the existing product. To achieve this, manual user testing was deployed by the team members.

### 7.2.2 Linting

To ensure consistency between source files and to remove certain classes of bugs, the ESLint<sup>7</sup> linting tool was used. This tool accepts a file containing rules that need to be enforced on the source code and analyses the code according to these rules. When a violation is found, it returns a warning stating what is wrong. This ensures that the code in the project stays consistent and that certain types of bugs are prevented by flagging dangerous practices.

### 7.2.3 Continuous Integration

A continuous integration (CI) service makes sure that a project runs on multiple machines without issues and that new code adheres to the desired standards. When a new feature is pushed to the GitLab remote repository, it is built by the CI. This ensures that the code does not have any dependencies that are available on the developer's machine but are not included in the build instructions.

Once the code has been built, the CI runs all tests (7.2.1) in the project to make sure that all tests pass and that the new code is up to the same standards as the rest of the project. When the CI fails to build or when a test fails, the CI fails the new feature branch and indicates that it shall not be merged into the master branch.

For this project, the CI was used to build the software and to run the test suite. Running the linter was not included in the CI process. Building and testing the application was relatively easy to achieve, but due to complications in the process of setting up the CI in combination

---

<sup>5</sup><https://jestjs.io/>

<sup>6</sup><https://sinonjs.org/>

<sup>7</sup><https://eslint.org/>



with the linter, the CI lacked the ability to run the linter when new code was pushed. Instead, the developer was responsible for checking the linter tool before pushing the branch to GitLab.

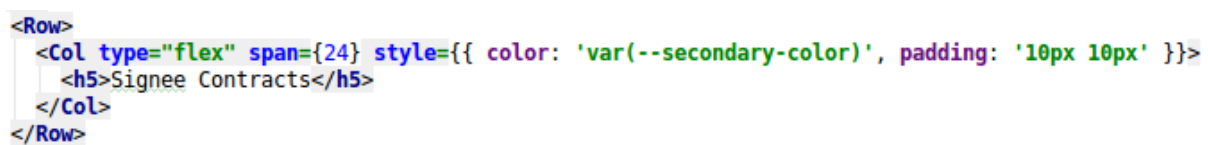
There were complications during the setup of the CI, causing it to take multiple days to get the system working. The used GitLab version at Lizard Global was outdated and likely contained a configuration issue, causing git not being able to clone a project using the http protocol. This restricted git to the usage of the ssh protocol when cloning the project. The issue was that the GitLab CI by default used http to clone a project and it was not possible to overwrite this default to use ssh instead. To solve this problem, the automatic cloning of the CI was disabled and a new stage was added to manually clone the project over ssh. In other words, the CI was partially disabled and code was injected to clone the project over the correct protocol. It took multiple days to first find out why the CI was failing to clone the project and then to implement the fix for the issue.

#### 7.2.4 Software Improvement Group

Halfway through the project, the developed code was submitted to the Software Improvement Group <sup>8</sup>(SIG) to receive feedback on the quality of the code. SIG analysed the code to look for issues with the maintainability of the code. The maintainability of code is often described as how easy it is to understand and modify existing code. When the maintainability of a piece of code is low, other developers tend to find it difficult to understand the code. Alternatively, making changes in one piece of code can then cause other pieces of code to stop working.

The feedback (in Dutch) that SIG provided has been included in subsection D.1, and although the code had a market-average maintainability of four out of five and a half stars, issues were found relating to code duplication and the size of methods. The analysis found that there were a number of methods that were significantly longer than the average method length. The feedback suggested to re-evaluate those methods to see if they had too many responsibilities and to break up those functions into multiple smaller ones. Code duplication was found among multiple files. In the case of code duplication, the same code is copied in multiple locations. It was advised to keep the amount of duplicated code as low as possible.

To improve the quality with respect to method length, a new rule was added to the ESLint code analysis tool in order to find all instances such that they could be resolved. This rule enforced that no method was allowed to be longer than 25 lines. This length was chosen as it caught the instances that SIG pointed out, along with a number of other methods. The issue with enforcing this rule was that the render methods, which provide the visual representation of the application in the form of HTML and CSS, were typically too long as they often include only one declarative statement per line. In other words, a render method often had the first line saying 'Create a box here', and the next line 'create a title in this box'. This issue was amplified by the fact that every statement to open an element required another statement to close the element, effectively doubling the amount of statements that was required to display an element. An example of this way of notation can be seen in Figure 4. Thus, lowering the maximum allowed number of lines per method would have resulted in rejection of a large amount of the render methods.



```
<Row>
  <Col type="flex" span={24} style={{ color: 'var(--secondary-color)', padding: '10px 10px' }}>
    <h5>Signee Contracts</h5>
  </Col>
</Row>
```

Figure 4: Example of a partial render method, illustrating the issue of lengthy code

---

<sup>8</sup><https://www.softwareimprovementgroup.com/>



Figure 5: Code duplication report of the master branch on 14-06-2019



Figure 6: Code duplication report after refactoring of the master branch on 14-06-2019

The issue with code duplication was solved in an unconventional way: Instead of refactoring the duplicated code into a single function, duplicates were critically analysed to see which elements they really needed, as the vast majority of these duplicates were methods to load Firebase into a page. When a page gets loaded, it can be connected to the Firebase database such that its internal state is in sync with the Firebase data. To do so, the Redux <sup>9</sup> state container was used. At the end of the pages that contained duplicate code, the same method for loading this state was pasted. To solve the duplication, each state loading method was evaluated to see which elements of the Firebase database were really needed for the component to work and all others were removed from the loader. The code duplication report before the refactor has been depicted in Figure 5. After critically evaluating the state loading methods, code duplication was significantly reduced, as can be seen in Figure 6.

To further improve the quality of the code, multiple extra rules were added to the ESLint tool in order to detect other common bad practices. The report after adding these rules can be found in Appendix E Figure 21. For example, the maximum nested depth of statements was restricted to two, as was the maximum allowed amount of chained callbacks. Furthermore, magic numbers (numbers which are not declared as a variable first) were prohibited and a new rule was enforced to make sure that all methods and classes had sufficient documentation. Appendix E Figure 22 shows the Eslint report after the found violations were successfully corrected.

A second submission to SIG was made at the end of the project, to verify that the feedback of the fist submission had been taken into account. The feedback on this second submission can be found in subsection D.2 and is written in Dutch. The analysis of the second submission

<sup>9</sup><https://redux.js.org/>

determined that the two issues found during the analysis of the first submission had been resolved while the amount of code had grown.

### 7.3 Integrating with blockchain

Integration of the blockchain took a significant amount of time, as the blockchain could only be run locally. The existing product already used a third party (NEM <sup>10</sup>) to run the blockchain. However, Lizard Global no longer had access to the remote blockchain, forcing development to use a simulator that ran locally. This brought along issues with the integration between Firebase and the blockchain. Namely, Firebase needed to write data to the blockchain, but was unable to reach it as it ran within the local network instead on the public internet. To solve this issue, a local Firebase emulator was set up that ran the functions that needed access to the blockchain. This way, the Firebase could be integrated with the blockchain. The Firebase functions were designed in such a way that they could be easily adapted to run on the real Firebase again once the blockchain becomes accessible.

### 7.4 Working with existing code

The project was implemented alongside the existing code base, which proved to be a challenge in certain cases. While most of the existing files remained unchanged, others had to be modified to allow for the introduction of new features and functionality. Where possible, the new code was moved to a separate file with some functions or a React component. In some cases, however, it was found sensible to further split up existing files first to make them more 'modular' before adding the new functionality. The original code base included some files of over 1500 lines, which made them difficult to understand and maintain. During the project, a number of these files have been dealt with when new functionality was to be introduced to these files. Examples of this are given later in section 8.

## 8 Results

In this section the results of the project will be discussed. A detailed description of the system that has been implemented will be given together with the difference between the initial design and final implementation. Additionally, an analysis of the implemented requirements, the final test coverage and code quality report and personal growth will be discussed.

### 8.1 Implemented system

This subsection describes each of the components in the system that has been implemented. Firstly, the reworked process of inviting a signee and signing a contract is explained. Next, functionality to update, download and delete personal data is discussed, and finally divergences from the initial design are listed and parts of the original code that have been broken up in smaller parts and adapted are described. An overview of all components that were added, removed or modified can be found in Appendix C Figure 20.

#### 8.1.1 Signing a contract

The status panel in the contract viewer allows the contract owner to invite a signee for a contract. He fills in the name, email address and mobile phone number of the signee. When the signee is submitted, a check on the email of the invitee is done to verify that this invitee has not yet signed the document or received an invitation for that contract, as the emails serves as identifier for the account.

---

<sup>10</sup><https://nem.io/>

If the invite passes the check, the signee is invited to sign the contract and a *signee\_invites* entry is created in the database (see Appendix A). Otherwise, the contract owner gets notified and the signee is not invited a second time.

Whenever a change occurs in the *signee\_invites* table, an automated check is conducted in Firebase to delete any invite entries that have existed for longer than 48 hours, because this invite contains the personal data as added by the contract owner. GDPR principle 5 (subsubsection 5.2.1) demands that this information is not stored any longer than required.

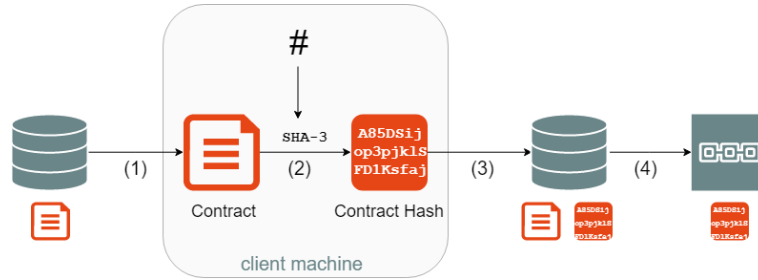


Figure 7: Process of sending a document to blockchain

Before the signee is able to sign, the contract is immutably stored in blockchain, as depicted in Figure 7. The contract stored in Firebase is requested by the client (1). SHA-3 is used to create a hash of the document (2), which the client stores in the database (3). The server then stores this hash in the blockchain (4) such that its state is retained indefinitely.

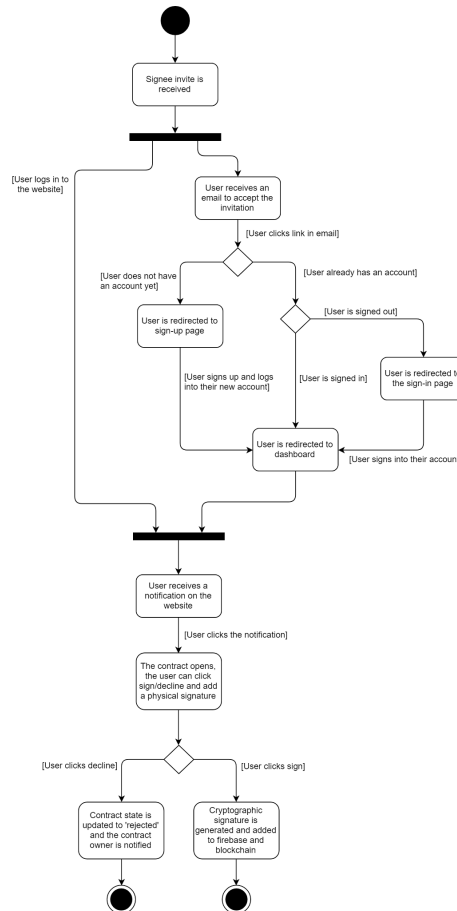


Figure 8: Activity diagram of the signee in the final system

The activity diagram of a signee being invited and signing a contract has been depicted in

Figure 8. When an invite is created, an email is automatically sent from the server, containing a URL that, when clicked, opens the ContractChain web page. If the user is not a known user in the system, he is redirected to the sign up page to create an account. The contract will be available for signing only after an account is created and the user is logged in.

If, on the other hand, an account is already registered on this user's email address, the user is either redirected to the dashboard page if he is already logged in, or the log in page otherwise.

Once the user is logged into the ContractChain environment, a notification is generated which displays the invite with this user's email address.

The user can also log in or sign up for an account without using the link in their email. In this case, the notification is generated in the same way.

When the user clicks *open* in the notification, the PDF view of the contract is loaded from the database (step (1) in Figure 10), with at the bottom a user interface element to sign or decline the contract. This element is depicted in Figure 9.

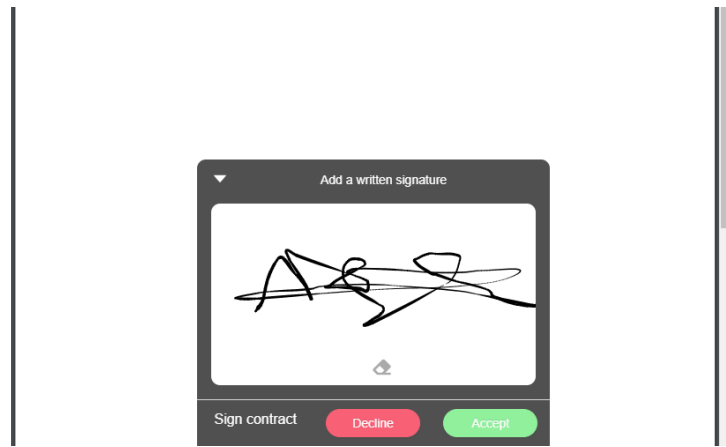


Figure 9: The user interface element for signing a contract

The user is also able to add a physical, written signature to the contract by drawing one with their mouse or on their touch screen.

If the signee declines the contract, the contract status is updated to 'rejected' and the contract owner and administrators are notified of the rejection. No further action is taken.

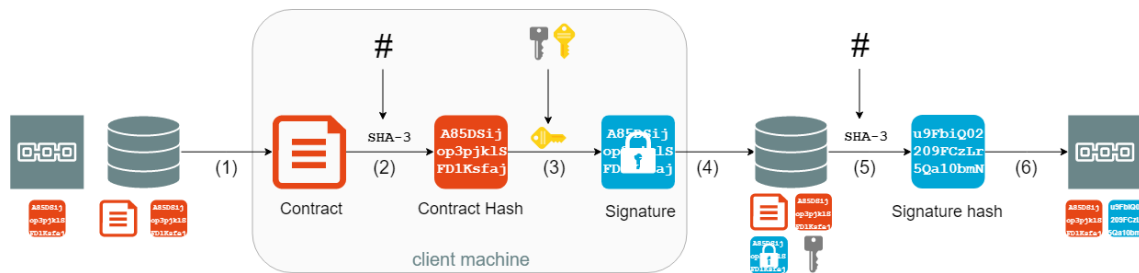


Figure 10: Process of signing a document and sending the signature to blockchain

If the signee accepts the contract, the signing process depicted in Figure 10 is started. First, a 512 bit SHA-3 hash of the document is constructed (2) and a cryptographic key-pair is generated. The private key is then used to sign the aforementioned document hash with OpenPGP (3). After signing the contract, both the private key as well as references to it are deleted from the memory. The signature object and the URL of the written signature are stored in the database (4), as well as the public key, which will be used with OpenPGP later down the line to verify the validity of the signature (Figure 12).

After the signature has been stored in the database, a call is made from the server to store the 512 bit SHA-3 hash of the signature (5) in the blockchain (6). Before it is sent to the blockchain, its validity is verified using the public key. Once it is stored, the signee's status is updated once more and he is notified by email that his signature is now immutably stored in the blockchain. If the server detects that the signature is not valid or if the signature could not be added to the blockchain, the status of the signing is set to 'failed' and the signee can try to sign the contract again.

A hash of the signature is stored in the blockchain, because the signature itself is too large to send to the blockchain. After signing the document hash, the signature is around 2300 bytes. Storing this would result in a transaction fee of 73 XEM <sup>11</sup>. Storing the 512 bit hash of the signature would only cost 17 XEM.



Figure 11: The user interface element for viewing a signature

After the contract is signed the signee is able to view his physical signature and verify his digital signature. The user interface element is similar to the one used for signing the contract and is depicted in Figure 11. Apart from the current status, the physical signature and the time stamp of the signature, an icon is displayed on the right side, by means of which the user is able to verify his signature. This verification is done in two steps. Figure 12 depicts the process of the verification on the server.

In the first step, the server has to verify that items stored in the database are the same as stored in the blockchain (Figure 12a). This can be done by retrieving the hashed signature and contract from the blockchain, and the non-hashed signature and contract from the database. Next, the server creates a hashed signature and hashed contract. Now the server can compare the hashes from the blockchain to the hashes of the database to see if they are equal. If the hashes are equal, the server knows that the items stored in his own database are the same as the ones added to the blockchain.

<sup>11</sup><https://nemproject.github.io/#transaction-fees>

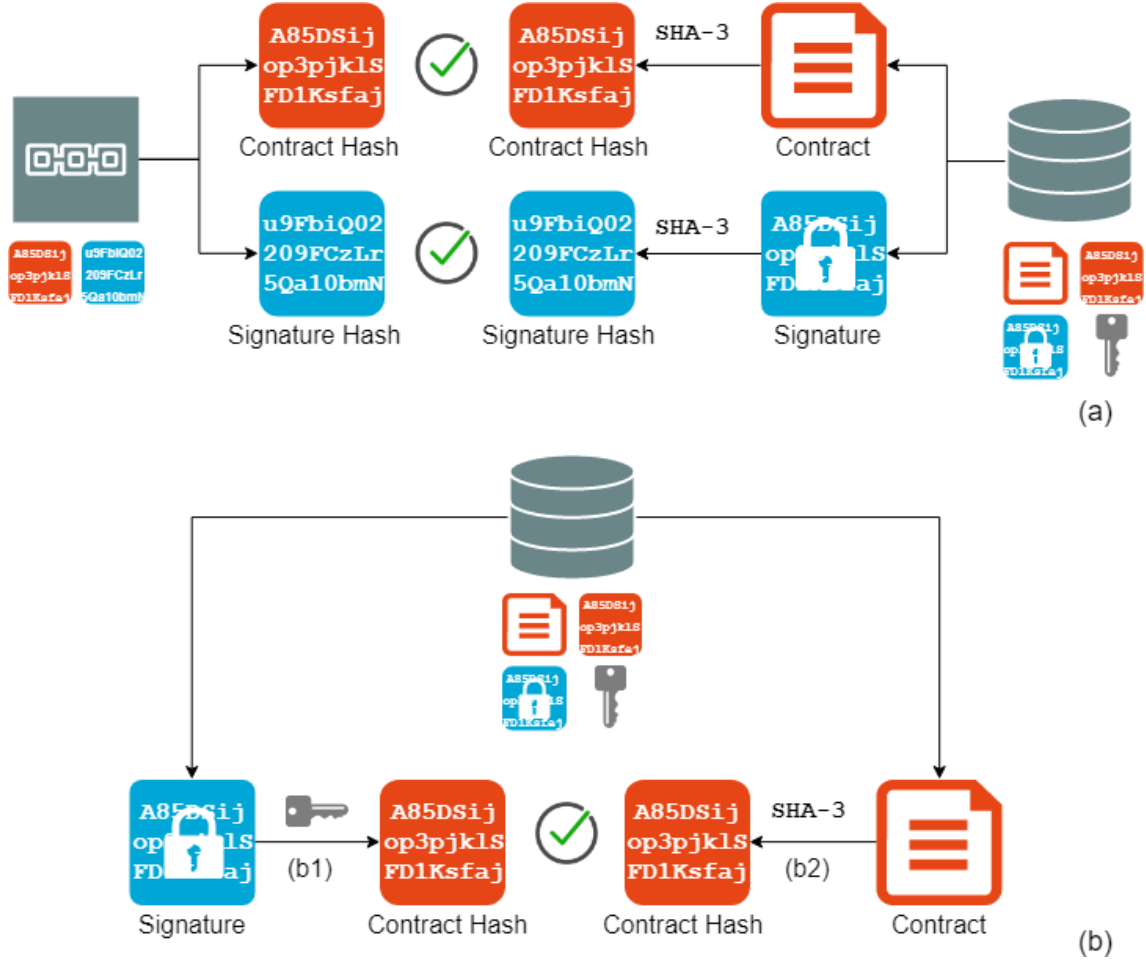


Figure 12: The signature verification process on the server

In the second step, the server checks the authenticity of the signature in the database (Figure 12b). The server retrieves both the signature and contract from the database. With the public key, the server is able to decrypt the signature and obtain the 512 bit SHA-3 document hash (b1). Next, the server creates a 512 bit SHA-3 hash from the contract (b2). The server is now able to check if the hashes are equal and thereby if the signature belongs to the contract. If the signature is invalid, the signee is not able to sign the contract again. A banner displaying that the signature is invalid will be displayed.

### 8.1.2 Updating information

In order to comply with GDPR, each user must be able to control his personal data (subsubsection 5.2.2). To solve these issues, a user panel was created, as is shown in Figure 13, from where a signee can manage his personal data:

- The user is able to update his first- and last-name, email, password and phone number. This solves the right of rectification, as the user is now able to update his data.
- The user is able to download his personal data. When he clicks the button, the server constructs a JSON object with the known first- and last-name, email, phone number and the name, content, physical signature and preview of the contracts that the user has signed, plus the public key and the signature used belonging to this contract. This solves the Right of portability.

- The user panel contains a table with the names and statuses of previous opened but not yet signed and signed contracts. This table is clickable and directs the signee to a page where the content, the signature status and if present the physical signature of the signed contract are displayed.

What is more, the first and last name, email and phone number known by the system are also displayed in the user panel. This solves the Right to access, as the signee is able to view all his personal data.

- The user panel has a button to stop processing data. When this button is clicked, ContractChain is not able to process the user's data to sign new contracts. When a the user tries to sign a contract, a pop-up is displayed with the message that his account is frozen. When another user tries to invite a signee with a frozen account, the user gets a message that the signee cannot be invited. This solves the right to object to and right to restrict processing of personal data.
- A signee is not able to delete his data, this is because his information is needed to retrace the signature to the signee. In other words, there is a legal reason to store this information in order to keep the contracts valid.

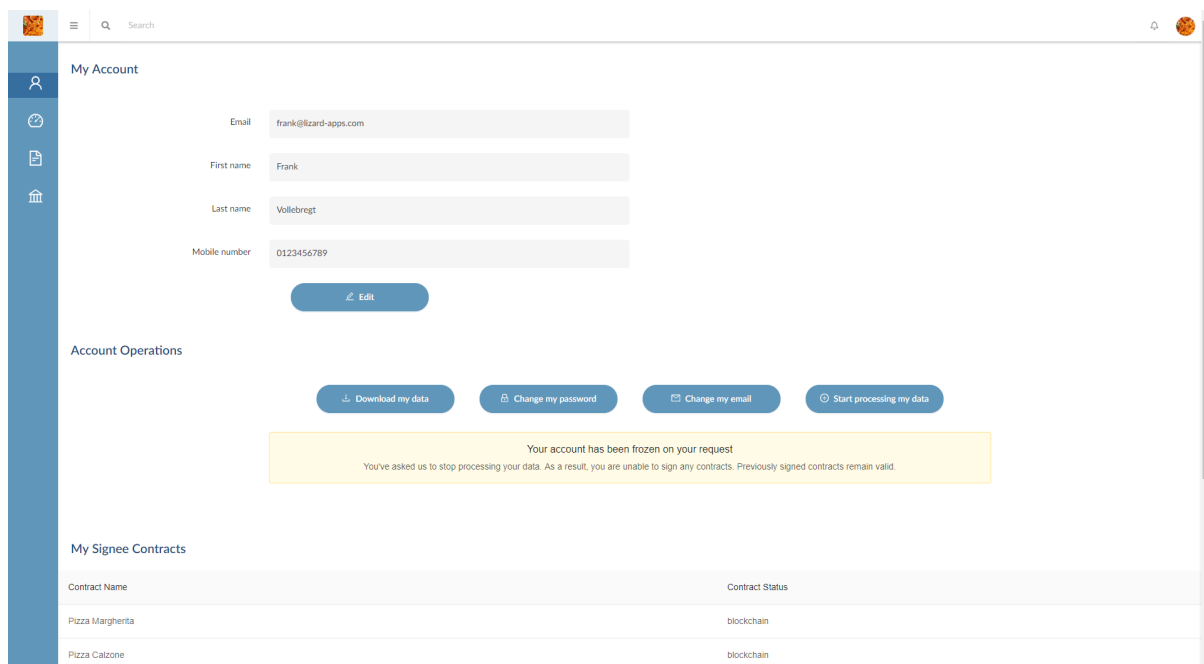


Figure 13: Screenshot of the implemented user panel

### 8.1.3 Deleting information

It could be the case that a contract contains the name of a signee. Therefore, when a contract is deleted by a user, all the instances of this contract need to be deleted. Together with the contract, all the signatures and public keys also need to be deleted.

When a contract is reviewed and ready for signing, a PDF version is stored in a Firebase storage bucket. In the same bucket a preview and possibly physical signatures will be stored. When a contract is deleted, these files should be deleted as well. This is achieved by putting a listener on the contract. Upon delete, a request to the bucket to delete all aforementioned files is sent. Because the signature and public key are fields of the contract, these items will be deleted together with the contract itself.



#### 8.1.4 Divergence from initial design

As indicated in 7.1.1, it was often needed to reevaluate the solution in mind during development. This led to alterations of the initial design. For instance, from the figures in appendix A and figures 2 and 8 can be seen that the design for both the signee invitation process and the database design are changed. The main reason to change the database design was to keep the database queries fast and efficient. When designing the database, there was no overview of what information was queried in which functions and in what way. These aspects were therefore not taken into account, which ultimately ended up in a number of necessary design changes.

The main reason for changing the invitation process was to make the process more user friendly. With the new design the user does not receive two emails, but is able to navigate to the contract directly after signing up by using the notification bell.

#### 8.1.5 Adapting existing code

In Figure 20 in Appendix C, it can be seen that some components on the client side are marked orange. This colour indicates which files contain functionality that was present in the initial version of ContractChain, but which is broken up into smaller components to assure better maintainability and extensibility. This subsection highlights for each case why the choice was made to extract the code from the parent.

**Notifications** In the original code, notifications were generated from multiple different places: Firstly, notifications were generated in the top-level *PageLayout* component, where database queries were performed to see if there were any new notifications, which was then indicated by the notification bubble near the bell icon. Secondly, notifications were generated in the *PendingContractList* component, which was loaded only after the notification drawer was opened and which also queried the database. Note that *PendingContractList*, despite the name, rendered *all* notifications, and not only those for contracts awaiting review. The above approach caused some unexpected behaviour where the bell icon would show a notification bubble while no notifications were present or vice versa. A schematic overview of this interaction can be seen in Figure 14.

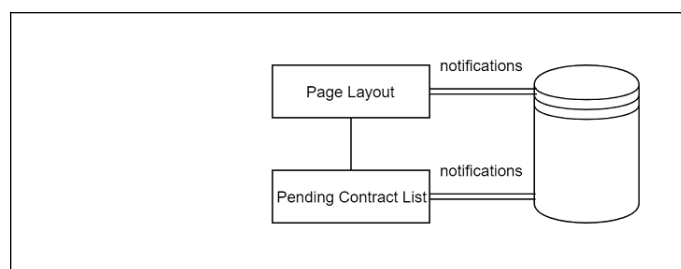


Figure 14: Notifications in the original code base

With over 650 lines, the *PendingContractList* file was difficult to understand and maintain. When a notification with a signee invite was to be added, it was decided to restructure the notification panel in such a way that it was built up from several simpler components, and minimise the database interaction. To achieve this, all responsibility to fetch notifications was moved to the *PageLayout* component. This component passes down the list of notifications to the *NotificationList* component, which renders the correct component from the *notifications* file depending on the *type* field of each notification. It has thus become relatively easy to add new types of notifications or change the style or layout of a certain notification type. Because the notification bell listens to the same list of notifications, desynchronisation in the notifications no longer occurs. An overview of the new notification structure is shown in Figure 15.

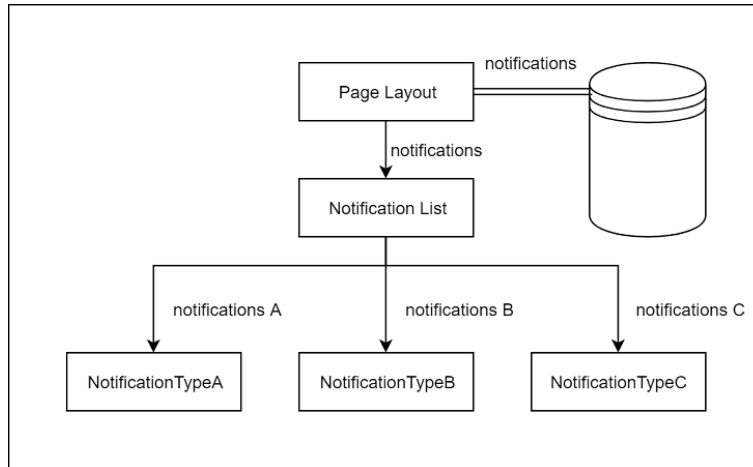


Figure 15: Notifications in the improved code

**Buttons in contract view** In the original code base, the buttons which a reviewer could use to 'accept' or 'decline' a contract were coded directly in the contract viewer file itself. When similar buttons for signees were to be added to the contract viewer, as well as a status message that was to be displayed to the user when a document was being signed or was signed, it was decided to move the review buttons, signee buttons and signee message all to separate files with React components, such that they were easily maintainable and linting and testing could be applied to them. This also serves to make the contract viewer file slightly easier to understand.

**Status panel and signee modal** From the status panel, the contract owner can open the signee modal to invite signees for the contract. The names and statuses of all invited signees are then displayed in a list at the bottom of this modal. In the original code, this list of signees is fetched from the signee modal, the code for which is included in the contract viewer file. When the contract's state is updated to 'waiting for signees', the same list is fetched again in the *StatusPanel* component. The code for fetching and rendering both lists of signees is virtually identical. In another effort to make the contract viewer component (named *ContractDetail*) more maintainable, we created a Firebase cloud function to fetch a list of signees, and introduced the *SigneeList* component. This component is then re-used in both the *StatusPanel* component and the newly added *SigneeModal* component. Once again, the layout or content of the signee list can now be changed with relative ease, and it will be changed wherever it is used.

**Sending emails from the server** Sending emails from the ContractChain server environment happens using an external API. In the original code, this was originally done from the signee functions file and the index file using a *sendEmail* file, where the email content was generated in the signee functions and index files, and passed to *sendEmail*, which made the API call. The two factor authentication (2fa) code email was sent via the same API, but this email was sent directly from the 2fa file. To make the code testable and more maintainable, the email functionality has been moved to a separate *EmailSender* file. The content for email messages is no longer generated in each separate file. Instead, a new *EmailGenerator* file is introduced, which can generate the email content, given the required information like the recipient's name and email address, as well as the contract name, 2fa code or company name. This is a more flexible approach that allows for easy extensibility and a larger degree of maintainability.

**Configuration files** The original code contained a significant amount of hard-coded elements like the URL on which the page was hosted, a number of API keys and default email addresses, in various places throughout the project. To increase adaptability, this code has been moved to a configuration file. This configuration file can then be modified to change, for instance, the URL on which the application is hosted. Moreover, it is considered good practice to keep all sensitive data (such as keys) in a separate file, instead of shipping them directly with the source code.

## 8.2 Testing and ESLint report

As described in subsection 7.2.1 and subsection 7.2.2, unit testing and the ESLint linter have been used to ensure the quality of the delivered product. The results of those efforts will be discussed next.

An average branch coverage of 100% was achieved by the unit tests, as can be seen in Figure 16. This percentage was achieved by a total of 37 test suites, containing a total of 308 tests.

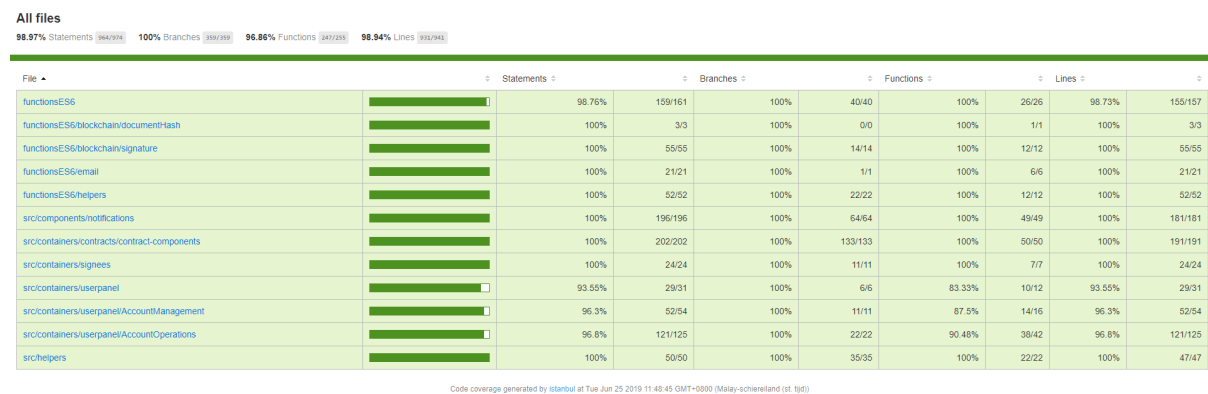


Figure 16: Test coverage achieved in the master branch on 25-06-2019

By using the ESLint linter, common issues in the source code were prevented. Appendix E Figure 22 shows the analysis of the tool on the source code. All issues that the tool reported were resolved, except for a small number of issues where the warning was suppressed. Suppression of warnings only happened when the developers had a good reason to do so, an example of this would be the ESLint reporting an error on a certain field name in the database. The advantage of adhering to the style guide did not outweighed the effort of refactoring the database to use a correct field name. It has to be noted that this field name already existed in the application before the project began.

## 8.3 Unimplemented requirements

During the implementation, the development team purposefully omitted functionality from the original requirements document (Appendix B). The redundant requirement was first discussed with the client before marking the requirements as 'not going to implement'. By request from the client, the list summing up the redundant requirements and their respective justifications has been moved to Appendix F.

## 8.4 Personal growth

Working in a real software development environment brought challenges and experiences that we had not encountered before during our study programme.

One of these experiences is to see how the team dynamics inside the software developing company worked. We had the chance to attend a meeting between the PO and PM where the PO was reporting the clients wishes. The distinction between the PO and PM during this meeting was evident. The PO was representing the wishes of the client and tried to convince the PM that these functionalities were important to include in the minimum viable product (MVP), while the PM was representing the development team and tried to convince the PO that the already implemented functionalities were sufficient for the MVP.

One of the challenges is working as an independent team alongside the existing product owner, product manager and lead developer. Part of this challenge arises from upgrading the existing software that is written by other developers, who had their own way for completing their projects. Their way was not to focus on the quality of the code, but rather to finish it in the least amount of time. This made integrating our software more challenging, as sometimes the existing code needed to be refactored to comply with the learnt software quality rules taught at TU-Delft.

Although we followed the course "Web and Database Technologies" during our bachelor, we have never implemented projects of considerable size using web-technologies. Most projects were implemented in Java. Thus, it was a challenge to learn the 'new' language and learn about its good practices using ESLint as code style checker.

Also, the existing code used a no-SQL database. This type was discouraged by TU-Delft, and thus this type would not have been the preferred choice. However, using it gave insight on the advantages and disadvantages of this type of database. An advantage would be that it is very simple to implement, as there is no obligation for consistency, which makes it useful for applications which do not need complicated queries from multiple tables. However, for big databases you need this consistency in order to prevent for example losing the data.

Another learning aspect was researching and applying the legislation concerning GDPR and the legal validity of digital signatures. During our bachelor, we were hardly ever required to take this into consideration. It was interesting how much GDPR legislation influenced the design choices of the project. For instance, the project before our research did not comply with GDPR. As a result, the whole signee invitation process needed to be redesigned. It would have been wise to identify relevant legislation and take them into consideration when designing the original system.

## 9 Ethical analysis

Along with the developed product arises an ethical implication concerning the usage of digital signatures. This section will discuss this issue.

### 9.1 Ethical issues concerning the means of signing

The developed application allows for two ways of signing a contract. For every signature, a cryptographic signature of the contract is constructed when the user clicks the "Sign" button. Besides this mandatory signature, a user can also draw his physical signature, which is then stored in the system.

Research has indicated that signing a contract by merely clicking a button makes people more likely to be dishonest [2]. The same research has indicated that people are more likely to be dishonest when they sign a document by clicking a button compared to signing the document by drawing their physical signature using the mouse of their computer. As the developed application only requires users to sign a contract by clicking a button and leaves the physical signature as an additional signature, ethical questions can be raised about this way of signing a document. Such questions consider whether it is ethical not to enforce a signing method which is known to increase honesty of users compared to the default method.

In this regard the project already improved the original method of signing a contract. In the original product, a contract could only be signed by clicking a button. Additionally, it is relatively easy to enforce the usage of a physical signature in the improved system, as the possibility has already been taken into account in the design, so only a few lines of code would have to be added.

## 10 Conclusion

Using agile development, the designed solution was implemented. The signee is able to sign a contract using an advanced e-signature. The e-signature together with the document hash are stored in a GDPR compliant manner in the blockchain. Moreover, the signees' personal data in the Firebase is stored in a GDPR compliant manner, as each signee can use his personal account to manage and control his data.

The code is tested thoroughly and a static code analysis tool is used to keep the code understandable. Using these tools and the feedback from the software improvement group, the quality of the code was kept high. This means that the developers of Lizard Global are able to continue on the delivered product.

Because of the missing tests of the original project, it was difficult to determine that no existing functionality was broken. But by user testing the code extensively and isolating our code as much as possible, it can be concluded that it is unlikely that the delivered product broke any existing functionality.

In conclusion, the criteria for the project to be a success as derived in subsection 3.3 are satisfied, and the project can be considered a success.

## 11 Discussion

This section will provide a discussion of the project from multiple points of view. First, all team members will give their personal assessment. Next, all team members together will provide a discussion on the implemented solution and on the performance of the team. Lastly, recommendations for Lizard Global will be provided to improve the ContractChain product.

### 11.1 Personal assessment

In this section, each team member will reflect on himself and his own performance during the project.

#### 11.1.1 Evaluation F.C.J. Vollebregt

The biggest challenge I faced during the project was understanding and working with the existing code base. As only very little up-to-date documentation was available, the only method to understand the code was to go through it step by step.

Out of the three of us, I was the most experienced with JavaScript, but this only helped me to a certain degree, as the React library changes much of the interaction between different files.

During the project I have tried to leverage React's features to the greatest extent possible by using small components to build up larger ones. Most notably I was responsible for modifying the signing process. This involved understanding and breaking up the existing components like the contract viewer and notification drawer, as well as adding the functionality for a physical and digital signature.

For me, this project has really stressed the importance of writing code that is well structured. During prototyping it is very easy to just mash together all different features, but in the long run this will hurt the efficiency of the development process, as a lot of time is spent on understanding

the code when it needs to be further modified. This is all theoretical, however, and in practice I am certain there will be countless more projects with similar challenges. In the future, I would try to get an overview of the project as early as possible (such as the diagrams in Appendix C), because this did help to understand the structure of the code. Secondly, rewriting the existing classes took more time and effort than I expected, and in hindsight rewriting the notification code was not strictly necessary (although it did significantly improve the code).

Similar to the bachelor seminar course, we had to conduct research on the topic at hand. In this case, we did not only write a report on this research, but we also actually used the outcomes. This made researching the problem and different solutions more interesting since during most previous projects of the study programme, we researched solutions as we encountered problems. Researching fitting solutions beforehand did ease the implementation process, so for future projects it is definitely advisable to do so to the extent possible. Working for an actual company was surprisingly similar to our university courses, though differences were present: Most significantly there was less direct supervision on the code quality, though I suspect this also greatly depends on the company. By employing the tools which we were taught to use during our study programme, we still maintained a high quality of code.

### **11.1.2 Evaluation H.M. Houwing**

The main challenge for me was that the project was written in JavaScript. I had not yet programmed or tested any significant projects using this language. However, I managed to pick it up rather quickly after doing a tutorial. In the end, I am satisfied with the additions I made to the product.

The biggest changes I was responsible for had to do with storing personal data in a GDPR compliant manner. This included implementing functionality for the signee to be able to change his/her email and password in the user panel. I also worked on deleting the users' personal data automatically. Furthermore, I spent a lot of time writing and reviewing the report.

For the functionality that allowed for changing the user's email, I ended up doing a lot of unnecessary work. This was caused by a lack of knowledge about what certain functions were doing in the background, which resulted in my solution not being compatible with the existing product and deleting part of already implemented functionality. Because the original project was not documented, it was hard to design a solution that fitted in beforehand. However, a part of the responsibility also lies with me. I could have done more research to the Firebase functions to find out what they did in the background. In the future, I would do more research to functions I use, such that implementing features that do not fit the solution can be avoided.

I learned a lot about how development is done in a small software development company. Because all the employees share the same work space, I was able to experience how the employees completed their tasks. It was interesting to see that in this small company, the development did not really use scrum. In the beginning of each week, the team did have meetings. But instead of daily meetings, the developers would just drop by for a quick question or update, which is similar to how we developed products during our bachelor's degree. It was also interesting to learn how the company managed the projects. Each project had one Project Owner, who managed contact with the client and represented their wishes. The Project Manager manages the budget and time of the project. We had the opportunity to attend the meeting for the start of the second developing stage. During this meeting, the Project Owner tried to convince the Project Manager to allocate new resources to implement and change functionalities, whilst the Project Manager tried to convince the project owner to accept a middle ground solution that would need less resources to implement.

From researching the legislation about digital signatures and GDPR, I learned that a programmer is obligated to implement an infrastructure for the user, such that the user is able to control all his personal data. During the project, the team and I figured out the best way to make this possible in ContractChain. I also learned how an individual is able to sign a digital

contract, such that his signature is legally binding. The two criteria are that the signature cannot be reproduced by someone else, and that the signature has to be connected to the document that is signed, such that an attacker is not able to copy paste the signature on another document, without it being detectable.

### 11.1.3 Evaluation M.J.W. van den Hoek

During this project, most of my tasks were concerned with either code quality assurance or with the GDPR aspect of the project. For the quality assurance, I have set up the CI and researched the proper frameworks to use in order to be able to test our code. I have also been concerned with preparing the project for the upload to SIG and with incorporating their feedback into the project. For the research report, I have done a significant part of the research into GDPR. Next, I applied this knowledge by implementing the initial version of the user panel, from where a user could modify and download his data and could request the application to stop processing his data.

I also completed multiple tasks which were not related to those two topics. For example, I set up the blockchain locally. As described in subsection 7.3, we were not able to run the externally hosted version of the blockchain, so a workaround was required to make it work locally, resulting in a refactor of the code base to make sure that all Firebase code that interacted with the blockchain could be ran from a locally running Firebase emulator.

When integrating the test frameworks into the existing code base, we were no longer able to deploy to Firebase, due to the Babel compiler not being able to work with the test framework. I have spent a significant portion of my time to work out how I could make them all work together without breaking the existing code base.

My biggest challenge in this project was to estimate the time that certain tasks would take, due to being unfamiliar with JavaScript and the existing infrastructure. For example, I thought that getting the blockchain to work would be a minor effort, as code to do so already existed in the project. However, I didn't realise that the unavailability of the NEM service would have such a rippling effect and that resolving the issue would take so much time.

One of my weak points during this project was my tendency to over-engineer the features that needed to be implemented. For some of the features, such as the user panel, I could have delivered a solution faster by making it less extended. However, for some features I am still happy that I went the extra mile. For example the incorporation of a configuration file to store all the settings that were spread out over the existing project.

In retrospect, I think this was a very interesting and educative project to be involved in, both in terms of research and gained work experience. During the research phase, I had to read the GDPR and translate it to requirements for the project. This required me to adapt to the style of writing, as I was not used to reading such documents. The requirements for a legally indisputable signature in combination with the requirements of Lizard Global's client brought along another learning point, namely the problems that one faces when intending to let the user sign a contract while no information can be stored at the user's device. Research into this topic revealed that there are common ways to do this, but that they all either require additional hardware, or that the user stores information on their device. The solution that we chose was an adaptation of an existing technique where the user is supposed to store his private key on his device.

I found the work experience that I gained during this project very valuable. The meeting with the project manager and project owner gave insight into the dynamics of a real software development team, as we had only enacted this dynamic within a student team during previous courses. Working alongside the team on an existing project also showed the project management and execution for an early-stage project at a startup. When working on the project, I got the impression that the software development methodology that had been used was sometimes more

focused on delivering a working product as soon as possible instead of on delivering a product that was of very high quality.

In conclusion, I think I have done quite a good job during this project, although there is still some room for improvement.

## **11.2 Group assessment**

To discuss the performance of the team and the implemented solutions, all team members together will evaluate their choices and performance during the project.

### **11.2.1 Team performance**

During the course of the project, the cooperation and performance of the team members was good. The agile development principles have not always been strictly adhered to, but due to the size of the team this did not result in any actual problems. We are aware, however, that a larger project, in which more people need to cooperate and communicate, demands an even more structured approach and more careful planning, such that everyone is able to keep track of what is going on.

Given more time to complete the project, more time could have been spent to learn and understand the existing system. This would have allowed us to design a better solution before it would be implemented. Due to constraints in available time and our limited knowledge on the underlying frameworks, our initial designs were not optimal. It would be preferable to learn these principles and subsequently create a complete solution design that is to be implemented.

We could also have invested more time and effort to assure the code quality. In the project, additional tools like a code duplication detection tool were only employed after the first round of feedback from SIG (Software Improvement Group). Still, some tools like the linter were already in place from the start of the project.

### **11.2.2 Implemented solution**

Because it was difficult to implement a qualified e-signature using a web-based application, the team chose to implement the advanced e-signature. However, a qualified e-signature would improve the legal implications notably.

## **11.3 Recommendations**

The team has multiple recommendations for Lizard Global to further improve their product in the future and to improve the quality of their code base.

### **11.3.1 Portability**

To give the signatures even more legal validity, it is recommended to step away from the requirement of portability. Making the signatures legally indisputable was complicated by the requirement that a user must be able to access the application from multiple devices. This prevented the solution from storing the user's private key on the device, thereby binding a user's signature to his device. By implementing a mobile app for signing the contracts, this extra layer of security could be incorporated into the product, strengthening the signatures. The fact that the application is already written in React makes it easier to develop a mobile application, as React allows to compile the application to multiple mobile platforms. This recommendation



does not suggest that the whole application should be developed for mobile platforms exclusively, it is merely recommended to disable the signing via the web interface and move this action to a dedicated mobile application.

### 11.3.2 GDPR compliance

This project has made the signing of the contracts GDPR compliant, it is however recommended to make the whole application GDPR compliant. To achieve this, it is advised to make sure that all users can delete their account from the user panel. During the project, accounts for signees were implemented for them to be able to exercise their GDPR rights. However, deleting an account requires changes over the whole project, as for example companies are also bound to the user who created them. Implementing the feature of deleting user accounts would mean than changes would reach far beyond the scope of this project. Therefore, this feature was not implemented. When downloading their personal information, signees should also be able to download the contracts they worked on instead of only those which they signed. Again, this feature fell out of scope as it did not concern the signing of documents.

### 11.3.3 NoSQL

Although the benefits of a NoSQL database for this project are recognised, it is recommended to move part of the data to an SQL database. The NoSQL database that is currently used allows for easy and fast retrieval of nested data objects such as the contract changes, signatures or public keys. However, it complicates the linking of data stored in different tables and requires data duplication for reasons of efficiency. Considering those drawbacks, it is recommended to migrate flat data such as user accounts and company accounts to an SQL database, while maintaining nested or large data such as the changes in a document or the signature objects in the NoSQL database. This way, the application can leverage the consistency promises that SQL makes for the data for which this matters while also leveraging the fast and efficient processing of large or nested data object that NoSQL offers.

### 11.3.4 Data duplication

Duplication of account information was found between multiple Lizard Global projects, this practice is discouraged due to the risk of introducing inconsistencies in the data. Users can log in using the Authentication-Core project, where their email was stored for the purpose of authentication. Besides this account for authentication, ContractChain also kept an account for each user, also containing their email address for the purpose of sending two factor authentication (2FA) emails. This structure worked as long as the user was not able to change their email address. When implementing the GDPR requirement of data rectification, issues with data consistency arose because it could happen that the email address was updated in one location but not in the other. To prevent such inconsistencies, it is recommended to keep data duplication to a minimum and to increase the integration between the different projects, such that ContractChain has easy access to the user data that Authentication-Core stores.

## References

- [1] Guido Bertoni et al. ‘On the indifferenciability of the sponge construction’. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2008, pp. 181–197.

- [2] Eileen Y. Chou. ‘What’s in a name? The toll e-signatures take on individual honesty’. In: *Journal of Experimental Social Psychology* 61 (2015), pp. 84–95. ISSN: 0022-1031. DOI: <https://doi.org/10.1016/j.jesp.2015.07.010>. URL: <http://www.sciencedirect.com/science/article/pii/S0022103115000979>.
- [3] European Commission. *Do we always have to delete personal data if a person asks?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/do-we-always-have-delete-personal-data-if-person-asks\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/do-we-always-have-delete-personal-data-if-person-asks_en) (visited on 26/04/2019).
- [4] European Commission. *What does data protection ‘by design’ and ‘by default’ mean?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/what-does-data-protection-design-and-default-mean\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/what-does-data-protection-design-and-default-mean_en) (visited on 26/04/2019).
- [5] European Commission. *What if somebody withdraws their consent?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/legal-grounds-processing-data/grounds-processing/what-if-somebody-withdraws-their-consent\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/legal-grounds-processing-data/grounds-processing/what-if-somebody-withdraws-their-consent_en) (visited on 26/04/2019).
- [6] European Commission. *What personal data and information can an individual access on request?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/what-personal-data-and-information-can-individual-access-request\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/what-personal-data-and-information-can-individual-access-request_en) (visited on 26/04/2019).
- [7] CORDIS. *The world’s largest secure collaboration suite*. URL: <https://cordis.europa.eu/project/rcn/213334/reporting/en> (visited on 30/04/2019).
- [8] Morris J Dworkin. *SHA-3 standard: Permutation-based hash and extendable-output functions*. Tech. rep. 2015.
- [9] European Commission. *Can I ask a company/organisation to stop processing my personal data?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rights-citizens/my-rights/can-i-ask-company-organisation-stop-processing-my-personal-data\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rights-citizens/my-rights/can-i-ask-company-organisation-stop-processing-my-personal-data_en) (visited on 26/04/2019).
- [10] European Commission. *Can individuals ask to have their data transferred to another organisation?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/can-individuals-ask-have-their-data-transferred-another-organisation\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/can-individuals-ask-have-their-data-transferred-another-organisation_en) (visited on 26/04/2019).
- [11] European Commission. *What happens if someone objects to my company processing their personal data?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/what-happens-if-someone-objects-my-company-processing-their-personal-data\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/dealing-citizens/what-happens-if-someone-objects-my-company-processing-their-personal-data_en) (visited on 26/04/2019).
- [12] European Commission. *When should I exercise my right to restriction of processing of my personal data?* URL: [https://ec.europa.eu/info/law/law-topic/data-protection/reform/rights-citizens/my-rights/when-should-i-exercise-my-right-restriction-processing-my-personal-data\\_en](https://ec.europa.eu/info/law/law-topic/data-protection/reform/rights-citizens/my-rights/when-should-i-exercise-my-right-restriction-processing-my-personal-data_en) (visited on 26/04/2019).
- [13] Hao Feng and Chan Choong Wah. ‘Private key generation from on-line handwritten signatures’. In: *Information Management & Computer Security* 10.4 (2002), pp. 159–164. DOI: 10.1108/09685220210436949. eprint: <https://doi.org/10.1108/09685220210436949>. URL: <https://doi.org/10.1108/09685220210436949>.
- [14] Mario Heiderich et al. *Pentest-Report OpenPGP.js 02.2014*. Feb. 2014. URL: [https://cure53.de/pentest-report\\_openpgpjs.pdf](https://cure53.de/pentest-report_openpgpjs.pdf) (visited on 30/04/2019).

- [15] Mario Lamberger and Florian Mendel. ‘Higher-Order Differential Attack on Reduced SHA-256.’ In: *IACR Cryptology ePrint Archive* 2011 (2011), p. 37.
- [16] Bob Larrivee. *E-Signatures in Europe: Understanding the legal requirements for proof of intent*. 2016. URL: [https://www.project-consult.de/files/AIIM\\_eSignatures\\_2016.pdf](https://www.project-consult.de/files/AIIM_eSignatures_2016.pdf) (visited on 29/04/2019).
- [17] *Node.js – a JavaScript runtime built on Chrome’s V8 JavaScript engine*. URL: <https://nodejs.org/> (visited on 26/04/2019).
- [18] *OpenPGP.js, OpenPGP JavaScript Implementation*. URL: <https://openpgpjs.org/> (visited on 30/04/2019).
- [19] European Parliament and Council of the European Union. ‘Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC’. In: *Official Journal of the European Union* L 119 (2016), pp. 1–88.
- [20] European Parliament and Council of the European Union. ‘Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive’. In: *Official Journal of the European Union* L 257 (2014), pp. 73–114.
- [21] Wouter Penard and Tim van Werkhoven. ‘On the secure hash algorithm family’. In: *Cryptography in Context* (2008), pp. 1–18.
- [22] *React – A JavaScript library for building user interfaces*. URL: <https://reactjs.org/> (visited on 26/04/2019).
- [23] Ronald L Rivest, Adi Shamir and Leonard Adleman. ‘A method for obtaining digital signatures and public-key cryptosystems’. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.
- [24] Yu Sasaki, Lei Wang and Kazumaro Aoki. ‘Preimage Attacks on 41-Step SHA-256 and 46-Step SHA-512.’ In: *IACR Cryptology ePrint Archive* 2009 (2009), p. 479.
- [25] Patrick Van Eecke. *ADOBE SIGN. Compliance with European electronic signatures legislation*. Dec. 2016. URL: <https://acrobat.adobe.com/content/dam/doc-cloud/en/pdfs/adobe-sign-eidas-compliance-uk.pdf> (visited on 29/04/2019).
- [26] Thomas Zefferer and Peter Teufl. ‘Leveraging the adoption of mobile eid and e-signature solutions in europe’. In: *International Conference on Electronic Government and the Information Systems Perspective*. Springer. 2015, pp. 86–100.

## A Database Structure Diagrams

Figure 17: Initial UML NoSQL database of the system. This figure has been removed on the request of Lizard Global

Figure 18: Final UML NoSQL database with the adjustments in order to get application GDPR compliant. This figure has been removed on the request of Lizard Global

## B Requirements in MoSCoW format

Must Have Signee signup:

- As a signee, when I get requested to sign a contract, the application must check if I already have an account
- As a signee, when I get requested to sign a contract and I don't have an account, the application should store my contact information for 48 hours
- As a signee, when I get requested to sign a contract and I don't have an account, I must get an email inviting me to create an account
- As a signee, when I click the link to create an account and it is no longer valid, I must be rejected
- As a signee, when I click the link to create an account and it is still valid, I must be able to create an account
- As a signee, when I click the link to create an account and it is still valid, the temporary entry of my contact data must be deleted

Contract invitation:

- As a signee, when I create an account and the data I provide matches with the data that was entered when I was invited, I should get an email inviting me to sign the contract.
- As a signee, when I get requested to sign a contract and I already have an account, I must receive an invitation mail to sign a document
- As a signee, when I clicked the link in the invitation, I must get a 2FA code via email
- As a signee, when I have an account, I must be able to sign a contract after receiving an invitation and completing the 2FA

Signing a contract:

- As a signee, when I sign a contract, a new RSA key-pair must be generated for me by my web browser
- As a signee, when I have a generated key-pair, I must be able to send the public key to the ContractChain (CC) via an HTTPS internet link
- As a CC, when a public key has been sent, I must be able to store this key linked to the signee who sent it
- As a signee, when I sign a contract and have my RSA key-pair, I must be able to create a 512 bit SHA-3 hash of the contract
- As a signee, when I have a hash of the document, I must be able to encrypt the hash with my private key to create a signature
- As a signee, when I have created the signature, I must be able to send it to the CC
- As the CC, when I receive a signature, I must be able to verify its validity
- As the CC, when I verify the validity of a signature, I must create a 512 bit SHA-3 hash of the contract

- As the CC, when I verify the validity of a signature, I must be able to retrieve the public key of the signee.
- As the CC, when I verify the validity of a signature, I must decrypt the signature using the signee's public key.
- As the CC, when I verify the validity of a signature, when my generated hash matches the one in the signature, I must update the status of this signee for this contract to signed
- As the CC, when I verify the validity of a signature, when my generated hash does not match the one in the signature, I must reject the signature.

## GDPR

- As a signee, when I am logged in, I must be able to view my personal profile, which shows my personal details
- As a signee, when I am logged in, I must be able to change my personal details (password, email and phone number)
- As a user, I must be able to request the application to stop processing my personal data
- As a user, when I have requested to stop processing my personal data, I must not be able to sign further contracts
- As a user, when I have requested to stop processing my personal data, the contracts I signed before the request must not be affected
- As a user, when I am logged in, I must be able to download my personal data
- As a user, when I download my personal data, I must get a copy of my account information, all my signatures and all the contracts I signed.
- As a user, when I download my personal data, I must receive it in a machine-readable format that allows me to import it into other applications.

## Should have

- As a signee, when I am logged in, I should have a list of contracts I have signed earlier and view them
- As a user, when I signed up, I should see a global write-up of the TOS clauses that are needed to extend to the new functionalities.
- As a company administrator, I should be able to request earlier requested signees to request a contract
- As a signee, when I have signed a contract with my unique RSA key-pair and the signature has been sent, the application should make sure to erase the private key from my browser memory
- As a signee, when I have signed the contract, I should see my signature next to the contract
- As a signee, when I create my account after an invitation to sign, I should not have to log in again with my new account.
- As a user, when I forgot my password, I should be able to reset it
- As a user, when I reset my password, I should get a reset link over email



- As a user, when I reset my password and I clicked the reset link, I should receive a 2FA over email
- As a user, when I enter the 2FA code, I should be able to reset my password

Could have

- As a signee, I could create a RSA key-pair once, linked to the account, and use that to verify the user's identity whenever he/she signs a contract
- As a signee, when I don't have an account, I could be able to sign up via home-page
- As a signee, I could get a notification to sign a contract if I am logged in already
- A way to not have an 'inactive' account as a signee, so that it can be activated by the signee at a later time
- A mobile application for signing documents where the RSA private key is stored locally
- All generated certificates could be signed by a qualified certificate belonging to the ContractChain, such that CC vouches for the authenticity of the user's certificate
- As a signee, I could choose to receive the 2FA code via SMS
- As a signee, when I clicked the link in the invitation email and I choose 2FA via SMS, I could get a 2FA code via SMS
- As a signee, when I send my public key to the server, my browser could establish an extra secure layer (on top of HTTPS) to prevent MitM attacks when HTTPS fails.
- The ability to sign with a picture or drawing of the actual (i.e. on paper) signature
- As a company administrator, when I am logged in, I could be able to add signee accounts to my company

Won't have

- As a user, when I sign-up, I will not get the TOS fully extended with all required clauses
- As a signee, I will not be able to sign a contract with a qualified advanced signature that I obtained from a third party qualified trust authority.
- As a signee, I will not be able to sign a contract using my biometric data
- As a user, I will not be able to remove any personal data from previously signed contracts
- As a user, I will not be able to choose my 2FA provider (no Google authenticator etc.)

## C Component Diagrams

Figure 19: Overview of components in the system of the initial code base. This figure has been removed on the request of Lizard Global

Figure 20: Overview of components in the system after all changes. This figure has been removed on the request of Lizard Global

## D Feedback SIG

The following sections contain excerpts of the emails from SIG containing the feedback on the project. Irrelevant parts such as instructions for the authors have been omitted.

### D.1 Feedback on first upload

Let tijdens het bekijken van de feedback op het volgende:

- Het is belangrijk om de feedback in de context van de huidige onderhoudbaarheid te zien. Als een project al relatief hoog scoort zijn de genoemde punten niet 'fout', maar aankopingspunten om een nog hogere score te behalen. In veel gevallen zullen dit marginale verbeteringen zijn, grote verbeteringen zijn immers moeilijk te behalen als de code al goed onderhoudbaar is.
- Voor de herkenning van testcode maken we gebruik van geautomatiseerde detectie. Dit werkt voor de gangbare technologieën en frameworks, maar het zou kunnen dat we jullie testcode hebben gemist. Laat het in dat geval vooral weten, maar we vragen hier ook om begrip dat het voor ons niet te doen is om voor elk groepje handmatig te kijken of er nog ergens testcode zit.
- Hetzelfde geldt voor libraries: als er voldaan wordt aan gangbare conventies worden die automatisch niet meegenomen tijdens de analyse, maar ook hier is het mogelijk dat we iets gemist hebben.

[Feedback]

De code van het systeem scoort 4.0 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Size en Duplication vanwege de lagere deelscores als mogelijke verbeterpunten.

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project:

- `downloadPersonalInformation.js:generateDataFile`
- `contractFunctions.js:verifySignature`

Bij Duplication wordt gekeken naar de hoeveelheid gedupliceerde code. We kijken hierbij ook naar de hoeveelheid redundantie, dus een duplicaat met tien kopieën zal voor de score sterker meetellen dan een duplicaat met twee kopieën. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om de hoeveelheid gedupliceerde code zo laag mogelijk te houden. Na verloop van tijd zal de gedupliceerde code moeten worden aangepast. Dit leidt niet alleen tot extra werk, aangezien op dat moment alle kopieën tegelijk moeten worden veranderd, maar is ook foutgevoelig omdat de kans bestaat dat één van de kopieën per ongeluk wordt vergeten.

Voorbeelden in jullie project:

- `src/containers/userpanel/AccountManagement/AccountManagement.js`  
en  
`src/containers/userpanel/UserPanel.js`

- `src/containers/userpanel/AccountOperations/AccountOperations.js`  
en  
`src/containers/userpanel/AccountOperations/DownloadPersonalData.js`

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

## D.2 Feedback on second upload

Hierbij ontvang je de resultaten van de hermeting van de door jou opgestuurde code. In de hermeting hebben we met name gekeken naar of/hoe de aanbevelingen van de vorige evaluatie zijn geïmplementeerd. Ook deze hermeting heeft het doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen.

We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een duidelijke verbetering in de deelscores te zien, zeker bij Unit Size.

Ook is het goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

## E Eslint reports

<b>ESLint Report</b>	
56 problems (56 errors, 0 warnings) - Generated on Tue Jun 25 2019 09:36:16 GMT+0800 (Malay-schiereiland (st. tijd))	
[+] E:\Vizard\Contract-Chain\functionsES6\contractFunctions.js	11 problems (11 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\functionsES6\deleteExpiredEntry.js	4 problems (4 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\functionsES6\downloadPersonalInformation.js	11 problems (11 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\functionsES6\email\emailGenerator.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\email\emailSender.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\userFunctions.js	5 problems (5 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\decideHelpers.js	1 problem (1 error, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\signees\SigneeInvites.js	6 problems (6 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountManagement\AccountInformationForm.js	2 problems (2 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountManagement\AccountManagement.js	1 problem (1 error, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\AccountOperations.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\ChangePassword.js	2 problems (2 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\DownloadPersonalData.js	2 problems (2 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\StopProcessingMyData.js	3 problems (3 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\WrapperChangePass.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\SignedContracts.js	2 problems (2 errors, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\UserPanel.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\WrapperUserpanel.js	1 problem (1 error, 0 warnings)
[+] E:\Vizard\Contract-Chain\src\helpers\signature.js	5 problems (5 errors, 0 warnings)

Figure 21: Eslint report after adding the new rules

## ESLint Report

0 problems - Generated on Tue Jun 25 2019 10:31:00 GMT+0800 (Malay-schiereiland (st. tijd))

[+] E:\Vizard\Contract-Chain\functionsES6\blockchain\documentHash\blockchainDocumentHashFunctions.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\blockchain\signature\blockchainInteraction.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\blockchain\signature\userInteraction.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\contractFunctions.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\deleteExpiredEntry.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\downloadPersonalInformation.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\email\emailGenerator.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\email\emailSender.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\firebase\tables.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\helpers\authHelper.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\helpers\blockHelperJSON.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\helpers\firebaseDAO.js	0 problems
[+] E:\Vizard\Contract-Chain\functionsES6\userFunctions.js	0 problems
[+] E:\Vizard\Contract-Chain\src\components\notifications\notificationHelpers.js	0 problems
[+] E:\Vizard\Contract-Chain\src\components\notifications\notificationList.js	0 problems
[+] E:\Vizard\Contract-Chain\src\components\notifications\notifications.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\buttonHelpers.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\decideHelpers.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\ReviewButtons.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\SignatureVerifier.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\SigneeButtons.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\SigneeList.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\SigneeMessage.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\contracts\contract-components\SigneeModal.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\signees\SigneeInvites.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountManagement\AccountInformationForm.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountManagement\AccountManagement.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\AccountOperations.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\ChangeEmail.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\ChangeEmailForm.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\ChangePassword.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\DownloadPersonalData.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\AccountOperations\StopProcessingMyData.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\SignedContracts.js	0 problems
[+] E:\Vizard\Contract-Chain\src\containers\userpanel\UserPanel.js	0 problems
[+] E:\Vizard\Contract-Chain\src\helpers\signature.js	0 problems
[+] E:\Vizard\Contract-Chain\src\helpers\timestampGetters.js	0 problems

Figure 22: Eslint report after correcting the all rules at the end of the project



## **F   Redundant requirements**

This appendix has been removed on the request of Lizard Global.

## G Info sheet

**Project name:** Digitally sign a contract and store it to the Blockchain with respect to the GDPR compliancy.

**Client organisation:** Lizard Apps Malaysian Development Centre Sdn. Bhd.

**Date presentation:** August 7th, 2019

**Description:**

The challenge of the project was to find a solution for signing a digital contract with a legally binding signature, and store it in a blockchain in a GDPR compliant manner. Our research concluded that an advanced e-signature enjoys more rights than a standard e-signature. Moreover, users have to be able to control their data stored in an application. Before implementing, the team designed the solution using UMLs and a MoSCoW requirements document. During the implementation, UMLs needed to be redesigned, as the designed solution did not fit the existing project. The product does now have a user panel, which allows signees to have control over their data and signees can sign contracts using a advanced e-signature. This product was tested using extensive user tests and unit tests that achieved 100 per cent branch coverage. The product will be further developed by Lizard Global. We recommended the client to make the whole application GDPR compliant.

**Martijn J.W. van den Hoek**

Interests: software development process, quality assurance, security

Contributions: **Research GDPR**, **quality assurance**, blockchain integration, user panel

**Frank C.J. Vollebregt**

Interests: software development process, user experience design

Contributions: **cryptographic signature process**, database refactor, **physical signature process**, user interface design

**Hendrik M. Houwing**

Interests: Team dynamics, scrum, encryption

Contributions: **update signee data**, access signee data, **GDPR compliant storage of data**

All team members contributed to writing the reports.

**Client** Jordi Röer

Affiliation: Lizard Global

**Coach** Dr. Zekeriya Erkin

Affiliation: Faculty of Electrical Engineering, Mathematics & Computer Science - Cyber Security group

**Contact information**

krijn98@outlook.com — martijn.vandenhoeck@protonmail.com — fcjvollebregt@gmail.com

The final report for this project can be found at: <http://repository.tudelft.nl>

## H Original project description

### **Digitally sign a contract and store it to the Blockchain with respect to the GDPR compliancy.**

Starting from 25th of May 2018, a new European law became active under the name GDPR - or General Data Protection Regulation. The GDPR-enforcement is an addition to the already active European privacy-law.

The introduction of the GDPR legislation has left a large impact on every organization. Organizations are required to inform and justify their customers and supervisor on how personal information/data is processed and stored. The organizations are responsible for the whole personal data processing process and are required to be aligned with the supplier and sub-suppliers. The data processing and alignment are captured in a processor agreement, which has to be defined for each company.

The ContractChain product helps companies with the building, signing, and maintaining these contracts. ContractChain provides control over the endless amount of contracts but also respects the privacy of others. The product offers: Through a pre-programmed set-up, we enable you to process processors create, manage, validate and update agreements. Similarities are digital (legally valid) to sign. We offer dashboard updates and real-time overview of the status of all (sub) processors in the chain. Events are registered in the blockchain to be 100% reliable guarantee. GDPR chain provides automatic messages on expiry of processor rights, data leaks and the request to delete data. GDPR Chain also asks the (sub) processors to periodically supply delivery conditions (docs such as privacy statement) upload.

The ContractChain product is currently being prepared for a second development phase. As part of this phase, LizardApps and the client are looking for a solution to store the signatures of the signee of a contract in the blockchain, making it non-disputable on legal terms. Next, both parties are also looking for functionality to draw the signatures on a digital contract. When a new signee is added to draw the signature, the whole contract (including previous signatures) should be renewed and stored in the blockchain.

The TU-Delft students will analyze suitable solutions for both functionalities. These solutions must respect the GDPR compliance currently active and the programming language currently used for ContractChain. The students will propose technical solutions that are feasible with respect to the client's budget and planning and LizardApps' technical capabilities. Once the solution has been decided, the students will work along with the development team in sprints of two weeks, setting SMART formulated goals and milestones, participating during daily stand-ups with the end goal to implement the chosen solution within the required timeframe.

### **Other information**

This assignment is already reserved. The students will work together on-site with the development team, located in Kuala Lumpur, Malaysia. LizardApps will help the students with their visa and with the residence. LizardApps will pay 900RM (Malaysian Ringgit) as a stipend. This is a standard stipend given to Dutch students having their internship at our office in Malaysia. English is mandatory as most of the team in Malaysia only speak English. The group size is limited to 4.