Smart Contractual Agreements Framework

Owen Joslin & William Joslin

Rochester Institute of Technology
Golisano College of Computing and Information Sciences
CSEC 659: Blockchain & Smart Contracts
Dr. Gahyun Park
Monday, March 29th, 2024

Introduction:

The manual processes of creating, negotiating, and executing contracts plague today's legal landscape, causing inefficiencies, delays, and high costs. These challenges have always existed; however, today's technological advancements offer transformative solutions. These advancements can leverage blockchain technology to streamline the contract lifecycle, producing a more efficient, secure, and transparent framework.

Background:

Creating and executing legal contracts is currently very labor-intensive, relying on numerous manual steps, most of which require intermediates. This process is outdated, involving significant delays that incur substantial fees for all parties involved. However, the emergence of blockchain technology offers a chance to change the legal contract process fundamentally.

The goals and objectives of Smart Contractual Agreements are multifaceted. The primary goal being: to develop a proof-of-concept that demonstrates the potential and feasibility of this technology and its possible integration into our legal processes. This project aims to achieve several objectives:

- Implement smart contract logic to accurately capture traditional legal agreements' terms and conditions and current relevant legal regulations and standards.
- Ensure a secure, immutable, and transparent record of contract execution, providing irrefutable evidence of contract terms and changes.
- Implement a secure authentication system to verify the identities of contract parties and prevent unauthorized access to all non-involved parties.
- Create a system allowing all parties to see if and when a contract is signed.
- Allow parties to access and view finalized contracts on the blockchain securely.
- Enable parties to verify the authenticity of contracts and confirm the signatures' identities via post-signature verification.
- Develop a user-friendly interface that allows for contract creation and signing.

This project's scope encompasses any agreement between two or more parties willing to have a public, immutable, and transparent contract assessable on the blockchain.

This project will achieve this proof-of-concept by utilizing Solidity smart contracts, the HardHat Ethereum Development Environment. It will also use GitHub version control to manage and maintain the codebase and enable further research by other developers.

The significance of Smart Contractual Agreements is vast:

- Efficiency & Cost Reduction: A practical implementation can eliminate human intervention and the reliance on intermediaries such as lawyers and notaries, dramatically reducing the cost of creating legal agreements between two parties.
- Transparency & Accountability: The contract creation process would be more transparent, as blockchain's immutable and tamper-proof nature ensures contracts cannot be retroactively altered. This fact enhances trust between parties and leads to higher levels of accountability.
- Disintermediation & Decentralization: By utilizing blockchain technology, parties can interact directly with each other and conduct decentralized agreements without relying on third-party intermediaries such as lawyers, notaries, or escrow agents.
- Legal Compliance & Risk Mitigation: Automating the contract process can help organizations comply with legal regulations and standards. By encoding legal conditions

into code-based smart contracts, parties can automate compliance checks and regulatory requirements, reducing the risk of contract breaches or disputes.

- Enhanced User Experience & Accessibility: Accessing and signing legal contracts via a comprehensive GUI allows parties with limited legal experience to create, negotiate, and sign contracts quickly. It also allows for remote signatures, enabling all parties to verify and sign anywhere with a network connection.
- Innovation & Competitive Advantage: Companies and individuals adopting an automated decentralized contract system can gain a competitive edge over competitors by modernizing their contract management practices.
- Global Reach & Scalability: This technology enabled organizations to execute contracts seamlessly across geographical boundaries and time zones, facilitating global business operations and international transactions.

Smart Contractual Agreements can provide a framework by utilizing the new blockchain technology, promising new benefits for organizations and individuals alike.

Methodology:

To start, some precursory terms and definitions need to be defined:

- *Terms:* This is the raw text string that defines the terms and conditions of the contract. (Appendix A)
- *Fields:* These are the fields that, per the user, are special data types defined in specific locations throughout the Terms string. These fields are stored in a Fields array. (Appendix B)
- *Front End:* The JavaScript front end that interacts with the Legal Contract Factory contract to deploy and manage Contract Templates and the Contract Packet Factory contract to deploy, manage, and sign Contract Packets.
- *Legal Contract:* A super class or parent smart contract that defines all the functions, variables, and parameters a Smart Contract Template uses. (Appendix C)
- *Legal Contract Factory:* A smart contract that creates child Contract Templates from the Legal Contract parent per the function calls of the Front End. (Appendix D)
- *Contract Template:* This is a child of the Legal Contract parent with populated terms and fields but not filled out. (Appendix E)
- *Contract Packet:* A super class or parent smart contract that defines all the functions, variables, and parameters a Smart Contract Template uses. (Appendix F)
- *Contract Packet Factory:* A smart contract that creates child Contract Packets from the Contract Packet parent per the function calls of the Front End. (Appendix G)
- *Packed:* When a Smart Contract packet is ‘packed,’ it is considered populated with the contract templates, like pages in a binder.
- *Signed:* A Contract Packet is considered ‘signed’ when all parties involved have signed the Contract Packet and the Status of the Contract Packet has been updated to either ‘Completed’ or ‘Active.’

The framework is designed to add any number of contract templates to a single contract packet and a Contract Packet can have any number of parties involved. This functionality enables users to deploy as many or as few contracts as they need while having any number of other users sign them all at once. This contract packet format reduces the need for all parties

involved to record all the addresses of potentially multiple documents scattered across the Ethereum Virtual Machine (EVM) when the packet provides one 'binder' address instead.

Creating A Contract Template:

However, getting to a packed contract packet requires many precursory steps. Since this is a framework, being flexible with almost any kind of user input was desirable. As such, to create a template, referring to Figure *1*, a user would incur the following steps:

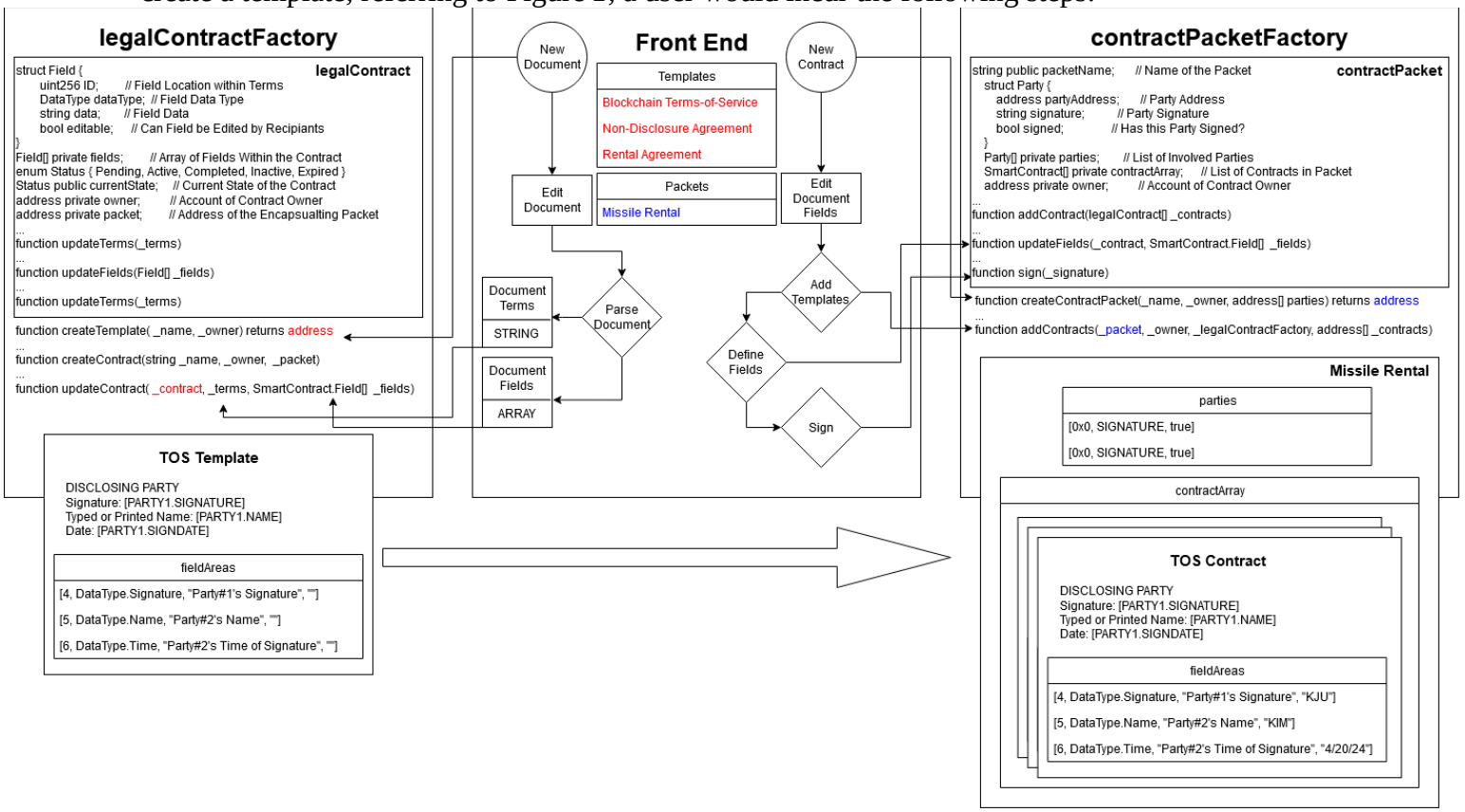

Figure 1. Deployment Flow Chart

1. In the Front End, a user with an account opens a new document. This triggers the Front End to call the 'createTemplate' function in Legal Contract Factory. This function creates a new child contract from the parent Legal Contract and populates the 'name' and 'owner' variables per the constructor. This creation is seen in Figure *1* and returns the address to the new contract.
2. In the 'Contract Editor,' the user would populate the document with their terms and conditions by dragging a fillable field from the left pane. These UI elements allow the user to place a field anywhere in the document, as seen in Figure *2*.
3. The user would then 'Save and Close' the template they created. This triggers the Front End to parse the text and field locations the user enters and create a Terms string (Appendix A) and Fields array (Appendix B).
    a. Deconstruct parsing (flowchart in Figure *1*) entails the fields' data type extraction and location within the document. These data types are enumerated where (and expandable to meet any data type needed):

`1=Signature, 2=Initials, 3=Date, 4=Time, 5=Name, 6=E-Mail, 7=Location, 8=Title, 9=Check-Box, 10=Text-Box`

b. These field types are then paired with their order number. (The number, from reading left to right, top to bottom, the parsing engine encounters.) As seen in Appendix B, the 'first' value in Appendix A is type 'date,' thus the first entry is `[1, 3, "", true]`, and the 'second' of type 'name,' thus `[2, 5, "", false]`, and so on. The last entry, 'boolean,' is set by the contract owner/deployer and determines whether or not the other parties involved can modify this value during the signing phase. (You wouldn't want another party changing your name in a legally binding contract!).

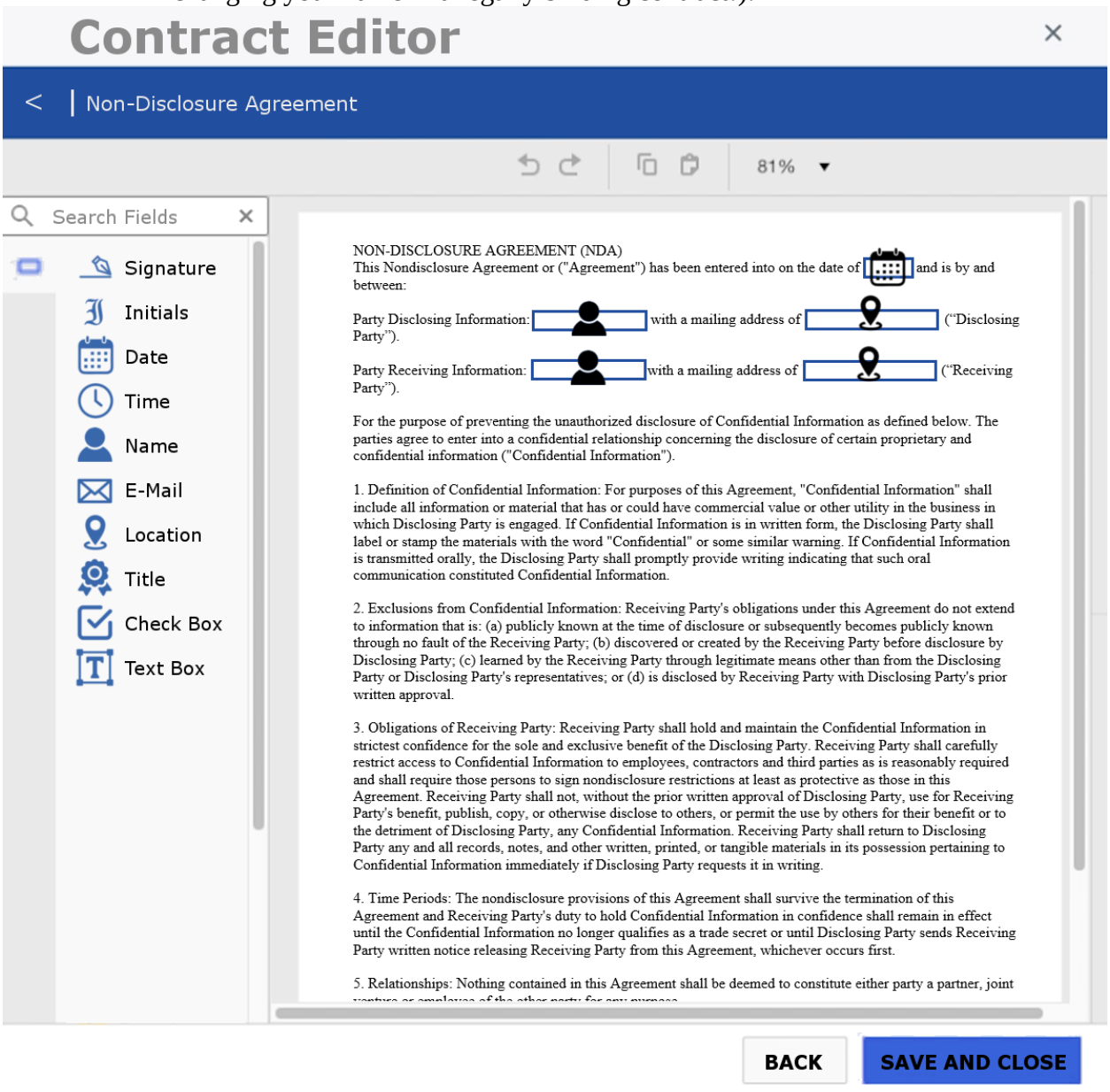

Figure 2. Contract Editor UI Design

4. After the user's terms and conditions have been deconstructed, the Front End calls the Legal Contract Factory's 'updateTerms' function. This function takes the newly created Contract Template address, parses Terms and Fields, and populates them in the contract by calling the contract's 'updateTerms' and 'updateFields' functions. Only after this step is the Contract Template usable in a Contract Packet.
5. After the Contract Template has been populated, the Front End saves the template address to the user's account ('Template' table in Figure *1*), allowing them to come back and change it if necessary. Doing so will increase the 'version' number.

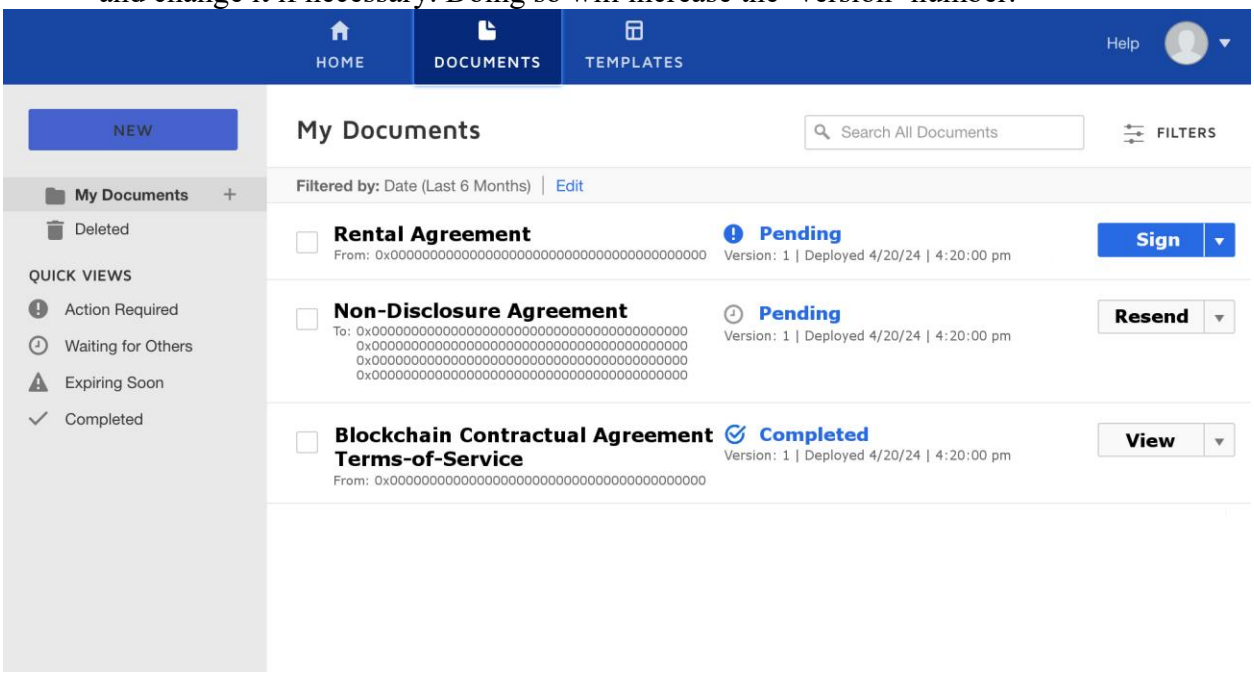

Figure 3. Contract Manager UI

Creating A Contract Packet:

After the user has made their Contract Template's, they may want to wait to deploy them. As such, the 'Templates' tab in Figure *3* allows users to view their templates. However, they may also want to deploy the template as a contract immediately. To do so, referring to Figure *1*, a user would incur these steps:

1. In the Front End, a user would enter the 'Templates' page (Figure *2*) and click 'Add to Packet'. A popup window would open and ask the user to select an open contract packet or to 'Create a New Packet.'
    a. If the user wants to add the contract to an existing Contract Packet instance, the Front End would call the Contract Packet Factory's 'addContract' function. this function calls the Legal Contract Factory's 'createContract' function that, like 'createTemplate' creates a new child of the Legal Contract parent. This address is returned and added to the end of the open packet's 'contractArray' array.
    b. If the user opts to create a new Contract Packet, the Front End is triggered to call the Contract Packet Factory's 'createContractPacket' function. The user is then prompted to add the name and the addresses of the parties involved (there is no limit). This function creates a new child contract of Contract Packet and populates

the variables ‘name,’ ‘owner,’ and ‘parties.’ It also automatically adds the service’s ‘TOS’ template to the first value [0] of ‘contractArray.’ This TOS contract instance is the ‘first’ page of the packet and enforces the legally binding nature of the signatures to the documents within the packet. The selected template then undergoes the same process as described in (a) above.

2. After the Contract Packet has been created, the Front End saves the returned address to the user’s account under the ‘packets’ table, per Figure *1*.
3. To populate the open fields of the templates within the Contract Packet, the user navigates to the ‘Documents’ page, where they can see all of their Sent or Received Contract Packets, their status, deployment date, and the parties involved (Figure *3*). Here, the user would select the new Contract Packet they named and select ‘Populate’.
4. A similar editor to the one used in Figure *2* would pop up, only this time the user cannot edit the text, only the fields (as mentioned before, if not the contract creator, only the fields allowed to be changed are fillable). Here, the Front End pulls in the templates referenced by addresses in the ‘contractArray’ and constructively parses them (Figure *4*). View ‘Retrieving Signed Contracts’ under ‘Additional Functionality’ for more details.
5. The deployer would then populate the fields and then click ‘Sign and Close’. This would trigger the Front End to call the Contract Packet’s ‘updateFields’ function. This function takes the now data-populated fields defined by the user and calls the ‘updateFields’ function for each Legal Contract instance in the ‘contractArray’. This populates the contract instances with the data for their fields. This updates the deployer’s ‘signature’ and ‘signed’ variable in the ‘parties’ array within the Contract Packet.
6. The Contract Packet address is sent to the other parties defined and the status of the Contract Packet is set to ‘Pending’ (Figure *3*). This Contract Packet instance is also added to the user’s ‘Documents’ web page.

Additional Functionality:

There are a few other core functionalities we needed to include in this framework. One being a way to retrieve and view any ‘Active’ or ‘Completed’ Contract Packet. And the other, being a way that received Contract Packets are signed by other parties who are not the deployer. These are their implementations:

- Signing Received Packets:
    1. Once a Contract Packet has been received by a party specified in its creation it will show up in that user’s ‘Documents’ tab as detailed in Figure *3*.
    2. The user will then select ‘Sign’ where, like in Figure *2*, an editor will appear where the user can enter any remaining information as determined by the boolean type of that field. They will then select ‘Sign and Close’ where, just like in step (5) above, the fields are updated per the ‘updateFields’ function that each contract instance in ‘contactArray’ inherits.
    3. Also like in step (5) above, the signer’s ‘signature’ and ‘signed’ variables are changed. If this is the last party to sign the document status is changed from ‘Pending’ to ‘Completed’ if there is no time duration set. Or to ‘Active’ if there was a time duration set. View ‘Signing Received Packets’ under ‘Additional Functionality’ for more details.
    4. As in step (6) above, the Contract Packet is also added to the party’s ‘Documents’ webpage.

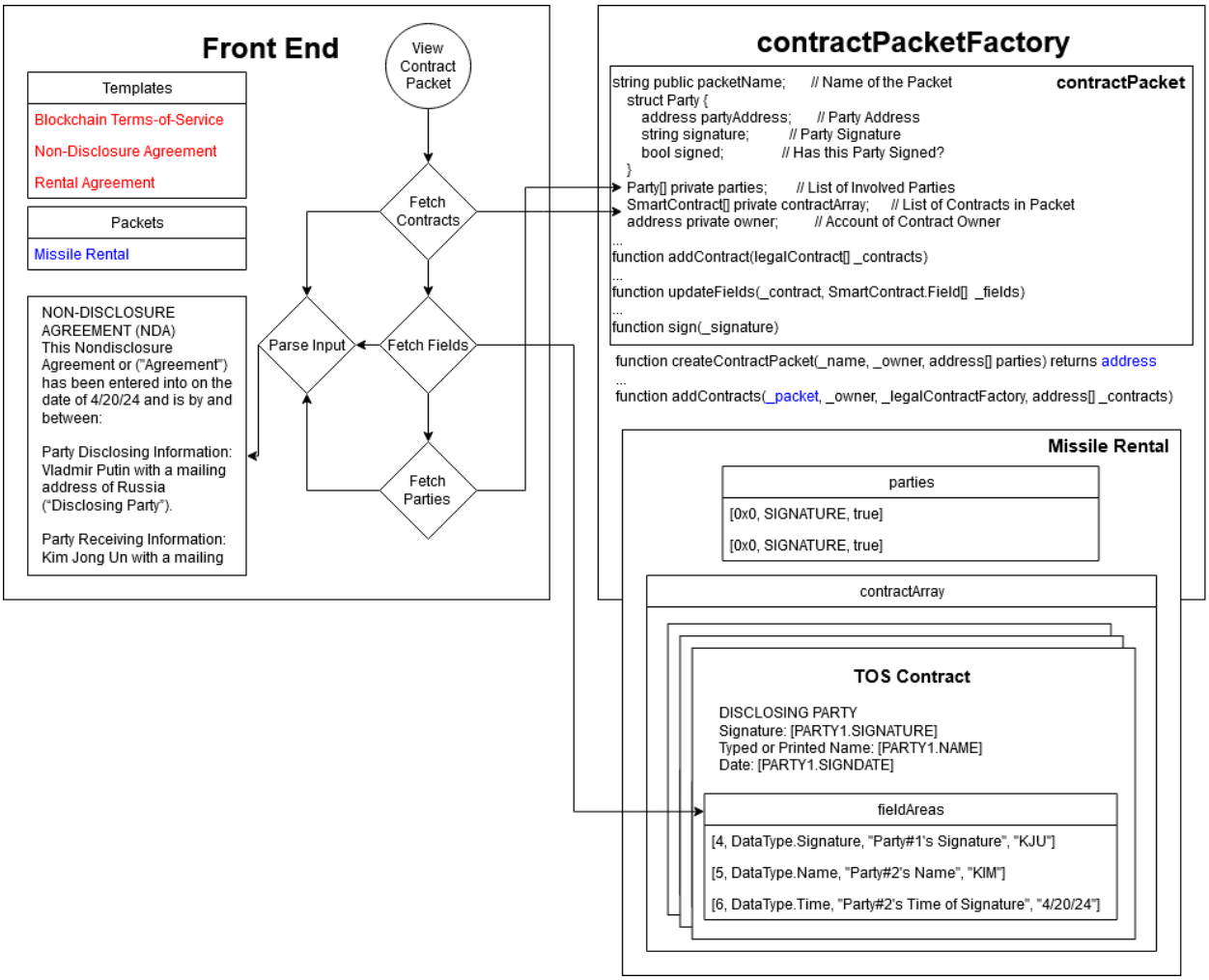

Figure 4. Contract Reconstruction Parsing

- Retrieving Signed Contracts:
    1. As described in Figure *4*, when a party wants to view the contracts within the Contract Packet binder instance, they are able to do so by selecting the 'View' button as detailed in Figure *3*.
    2. This triggers the Front End to fetch the contacts in the 'contractsArray' and constructively parse their Terms and Fields together to display the full human-readable contract. The Front End also needs to pull and parse the signature data from the 'parties' array to fill the data type 1 or 'Signature'. In this case using the signer's address since, as mentioned in future work, tying a digital signature to a person can be done through other means and was out of the scope for this project; among other things detailed below.

Future Work:

While this project provided much insight into how Smart Contractual Agreements may be implemented, there is much room for future development.

Front-end development would enhance the user experience and be critical to usability. Using options such as JavaScript, TypeScript, React, Vue, or Angular would give users an appealing interface that would show detailed information about each contract, such as document status, sender address, and deploy data. This interface would also include an intuitive contract editor that allows users to pull fields into the contract, as seen above in Figure *2* and Figure *3*. Adding a front end would also allow for integration into current platforms. ESC 725 Sovereign Identity for signature would offer promising paths for enchanting the security and authenticity of digital signatures within Smart Contractual Agreements. ESC 725 would also provide a multi-factor authentication process that can be implemented as an access control mechanism. This framework also presents a need to reduce the cost of contract deployment. Possible solutions involve integrating off-chain storage of non-sensitive contract data and documents or utilizing on-chain uniform resource identifier schemes to reference off-chain storage locations. This solution could also use cost-effective storage solutions such as decentralized storage networks or InterPlanetary File System integration. Future work may also include using RSA or CRYSTALS-KYBER keying to encrypt contracts with public keys, only allowing authorized parties to decrypt and view a contract on the blockchain. Integrating CRYSTALS-KYBER keying would also provide post-quantum resistance to contracts retroactively sorted on the blockchain. Finally, adding a marketplace for contract templates would allow for template sharing and collaboration among contract parties. These future works can enable Smart Contractual Agreements to grow into a strong, robust, user-friendly, and cost-effective platform for enhancing the contract management processes.

Appendix A

This is an example of what Terms may look like:

NON-DISCLOSURE AGREEMENT (NDA)
This Nondisclosure Agreement or ("Agreement") has been entered into on the date of **[1.DATE]** and is by and between:

Party Disclosing Information: **[2.NAME]** with a mailing address of **[3.ADDRESS]** ("Disclosing Party").

Party Receiving Information: **[4.NAME]** with a mailing address of **[5.ADDRESS]** ("Receiving Party").

For the purpose of preventing the unauthorized disclosure of Confidential Information as defined below. The parties agree to enter into a confidential relationship concerning the disclosure of certain proprietary and confidential information ("Confidential Information").

1. Definition of Confidential Information: For purposes of this Agreement, "Confidential Information" shall include all information or material that has or could have commercial value or other utility in the business in which Disclosing Party is engaged. If Confidential Information is in written form, the Disclosing Party shall label or stamp the materials with the word "Confidential" or some similar warning. If Confidential Information is transmitted orally, the Disclosing Party shall promptly provide writing indicating that such oral communication constituted Confidential Information.

2. Exclusions from Confidential Information: Receiving Party's obligations under this Agreement do not extend to information that is: (a) publicly known at the time of disclosure or subsequently becomes publicly known through no fault of the Receiving Party; (b) discovered or created by the Receiving Party before disclosure by Disclosing Party; (c) learned by the Receiving Party through legitimate means other than from the Disclosing Party or Disclosing Party's representatives; or (d) is disclosed by Receiving Party with Disclosing Party's prior written approval.

3. Obligations of Receiving Party: Receiving Party shall hold and maintain the Confidential Information in strictest confidence for the sole and exclusive benefit of the Disclosing Party. Receiving Party shall carefully restrict access to Confidential Information to employees, contractors and third parties as is reasonably required and shall require those

persons to sign nondisclosure restrictions at least as protective as those in this Agreement. Receiving Party shall not, without the prior written approval of Disclosing Party, use for Receiving Party's benefit, publish, copy, or otherwise disclose to others, or permit the use by others for their benefit or to the detriment of Disclosing Party, any Confidential Information. Receiving Party shall return to Disclosing Party any and all records, notes, and other written, printed, or tangible materials in its possession pertaining to Confidential Information immediately if Disclosing Party requests it in writing.

4. Time Periods: The nondisclosure provisions of this Agreement shall survive the termination of this Agreement and Receiving Party's duty to hold Confidential Information in confidence shall remain in effect until the Confidential Information no longer qualifies as a trade secret or until Disclosing Party sends Receiving Party written notice releasing Receiving Party from this Agreement, whichever occurs first.

5. Relationships: Nothing contained in this Agreement shall be deemed to constitute either party a partner, joint venture or employee of the other party for any purpose.

6. Severability: If a court finds any provision of this Agreement invalid or unenforceable, the remainder of this Agreement shall be interpreted so as best to affect the intent of the parties.

7. Integration: This Agreement expresses the complete understanding of the parties with respect to the subject matter and supersedes all prior proposals, agreements, representations, and understandings. This Agreement may not be amended except in writing signed by both parties.

8. Waiver: The failure to exercise any right provided in this Agreement shall not be a waiver of prior or subsequent rights.

9. Notice of Immunity: Employee is provided notice that an individual shall not be held criminally or civilly liable under any federal or state trade secret law for the disclosure of a trade secret that is made (i) in confidence to a federal, state, or local government official, either directly or indirectly, or to an attorney; and (ii) solely for the purpose of reporting or investigating a suspected violation of law; or is made in a complaint or other document filed in a lawsuit or other proceeding, if such filing is made under seal. An individual who

files a lawsuit for retaliation by an employer for reporting a suspected violation of law may disclose the trade secret to the attorney of the individual and use the trade secret information in the court proceeding, if the individual (i) files any document containing the trade secret under seal; and (ii) does not disclose the trade secret, except pursuant to court order.

This Agreement and each party's obligations shall be binding on the representatives, assigns and successors of such party. Each party has signed this Agreement through its authorized representative.

DISCLOSING PARTY
Signature: **[6.SIGNATURE]**
Typed or Printed Name: **[7.NAME]** Date: **[8.DATE]**

RECEIVING PARTY
Signature: **[9.SIGNATURE]**
Typed or Printed Name: **[10.NAME]** Date: **[11.DATE]**

Appendix B

This is an example of what template fields may look like:

```
FIELDS[ [1, 3, “” true], [2, 5, “”, false], [3, 7, “”, false], [4, 5, “” true], [5, 7, “” true], [6, 1, “”, false], [7, 5, “”, false], [8, 3, “”, false], [9, 1, “”, true], [10, 5, “” true], [11, 3, “” true] ]
```

Appendix C

Refer to: https://github.com/Blockchain-2235/project-william_owen/blob/main/Project%20Code/contracts/SmartContract.sol

Appendix D

Refer to: https://github.com/Blockchain-2235/project-william_owen/blob/main/Project%20Code/contracts/SmartContractFactory.sol

Appendix E

This is an example of a Contract Template:

Appendix F

Refer to: `https://github.com/Blockchain-2235/project-william_owen/blob/main/Project%20Code/contracts/SmartContractPacket.sol`

Appendix G

Refer to: `https://github.com/Blockchain-2235/project-william_owen/blob/main/Project%20Code/contracts/SmartContractPacketFactory.sol`