Proof-of-Concept Medical Device Complaint Tracking dApp

Katrina Siegfried, CSCI5240, Spring 2023

1 Background

1.1 Tracking Medical Device Complaints Isn't Just a Good Idea, It's Required

Under the FDA, there is very strict guidance around complaint handling for medical devices as outlined by 21 CFR 820.198. Medical device complaints must be processed in a timely manner, and with an auditable trail, to ensure that all events are investigated and can be reported to the FDA if needed. Most companies have a quality management system (QMS) which is structured to support compliance for medical device complaint reporting. These complaints are also consumed by other portions of the business outside of the quality compliance groups, such as

- Integration in a product backlog as a bug or feature enhancement request.
- For service teams to have up-to-date information and tracking on customer issues and for scheduling service.
- For tagging inventory systems to support regulatory restrictions or awareness of product issues.
- Integration with the design history file (DHF) of a product.
- Customer-facing personnel may log observations or information outside of complaints which also needs to be tracked.

In many instances, the software programs used in the applications listed above likely do not easily interface with one another. This causes issues where the information from a customer complaint is not consistently distributed, updated, and tracked across all different domains, resulting in disrupted information flow.

Additionally, these traditional database programs used for defect tracking are expensive and proprietary. This makes developing interfaces between them and all the other programs used across the business difficult if not impossible as it requires an agreed interface between many disparate programs. It also serves as a cost barrier for medical device startups.

1.2 Requirements for Medical Device Complaint Tracking

To ensure a complaint tracking program meets the standards set forth in FDA 21 CFR 820.198 which outlines the regulations for medical device complaint tracking, the following requirements must be met.

- All complaints must be stored indefinitely and are subject to audit.
- All complaint initial filings, addendums, and closures shall be documented.
- Complaints in the system shall contain the following information at minimum:
    - Summary (Title)
    - Description
    - Site of complaint
    - Severity
    - Corrective action (if applicable)
    - Product
- The system shall allow linking of complaints, addendums, and closures.
- The system shall allow retrieval of a complaint given an ID.
- The system shall allow retrieval of all linked complaints given a complaint ID.
- Complaint records should be protected from modification after the initial filing.
- The system shall be designed such that the complaint records are stored in a way that mitigates data loss.

1.3 Current Solutions for Medical Device Complaint Tracking

Current solutions are traditional database applications. These applications often come with hefty licensing fees up to tens of thousands of dollars per year minimum, require knowledgeable IT staff to maintain and secure a database either locally or in the cloud, and don't interface well with other commonly used applications[1]. These challenges can be barriers for small upstarts in the medical device field, forcing startups to rely on acquisition for penetration into the market.

The most common applications used for medical device complaint tracking include Windchill, ServiceMax, Zendesk, Jira Service Management, and Qualtrics[2].

1.4 What Ethereum and Web3 Can Bring to Complaint Tracking

The rise of Ethereum and Web3 provides the perfect opportunity for novel applications which require immutable record keeping[3]. The combination of cryptography and the blockchain hashing process guarantee that the records on the chain cannot be modified or deleted, ensuring complete data integrity and a simplified means of auditing – both critical in medical device complaint reporting.

Rather than hosting a critical database on a single server and having to employ a knowledgeable IT group to consciously mirror or replicate the data, Web3 and Ethereum allow the complaint ledger to exist in a distributed fashion across many different systems. If a node is taken out, the remaining nodes contain enough information to restore any missing information[4].

The open-source nature of Ethereum and Web3 set a precedent for any dApps deployed on the network. Hosting a project that runs on open-source code on an open-source platform further serves to decentralize risks and opportunities, including the discovery of bugs, improvements around application security, and feature enhancements[5]. Developing a program which is required by federal regulation that is open source will encourage the development of the program to move towards effectively meeting the needs of the regulation rather than towards a model which benefits the developers.

Limitations around the scalability of Ethereum are conveniently subverted by use of an Ethereum variant which would be used in a semi-private permissioned-based ledger system for medical device complaints, like the JPM Coin project by JPMorgan[6]. If running on a separate chain, the complaint system need not be burdened by the ebb and flow of unrelated Ethereum transactions.

1.5 Project Objective

The objective of this project is to provide a proof-of-concept for an open-sourced dApp and a foundation for the development of a set of open API's for common consuming programs which is designed to meet the requirements for medical device complaint handling by the FDA. This foundation of well-defined documents and proof-of-concept dApp will provide an artifact to drive further open-sourced work and discussion on innovations in the medical device community.

This project proposes an open-source, FDA compliant, secure, medical device complaint management system powered by the concept of the decentralized ledger as a decentralized application (dApp).

dApp Implementation

This proof-of-concept dApp was developed using several open-source and readily available technologies. The entirety of the code, information about the application, and all related resources can be found at https://github.com/siegfrkn/complaint-tracking-dapp.

[1] Common Medical Device Complaint Problems
[2] Best Complaint Management Software
[3] Ethereum Whitepaper
[4] Ethereum Nodes and Clients
[5] Open-Source Wikipedia
[6] JPM Coin

1.6 ARCHITECTURE

The application was developed in the Truffle Development Suite, which is a complete set of tools that allow rapid development and local deployment of Web3 dApps. Figure 1 shows the overall architecture and tech stack used in development and deployment.

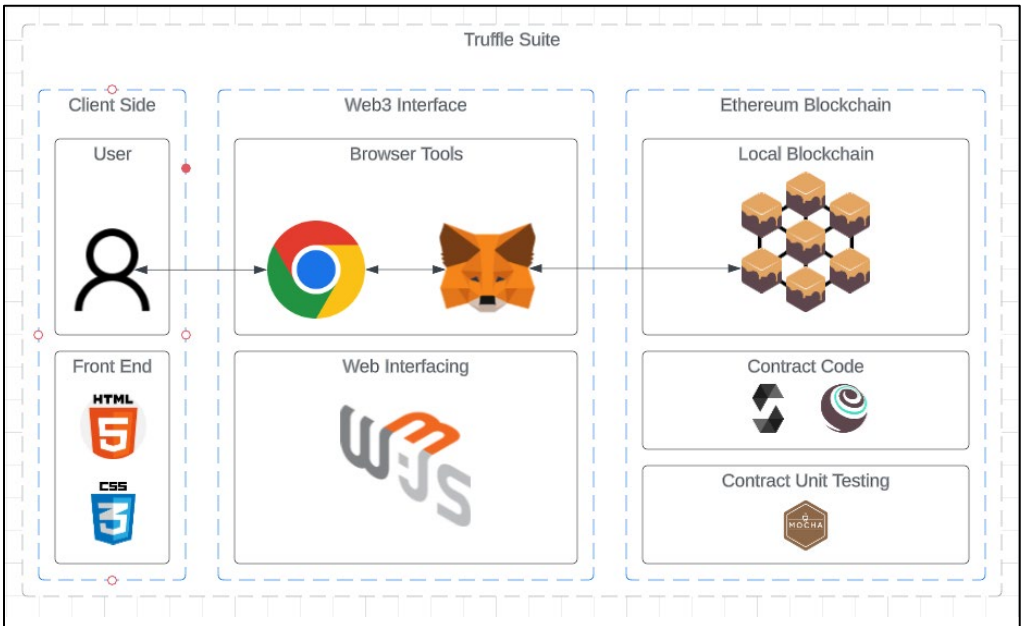

Figure 1: The architecture of the complaint-tracking-dapp.

1.6.1 Dapp Components

The complaint tracking dApp has three primary components – the Ethereum blockchain, the Web3 interface, and the client-side UI. Figure 2 shows a screenshot of the user interface.

1.6.1.1 Ethereum Blockchain

The Ethereum Blockchain makes up the "back end" of the dApp. In this proof-of-concept, a local Ethereum blockchain is generated and hosted by the Ganache development application which is hosted locally. The contract was written in the Solidity language which is "a statically typed curly-braces programming language designed for developing smart contracts that run on Ethereum.[7]" This dApp was written and deployed using the Truffle development framework which simplifies and expedites development.

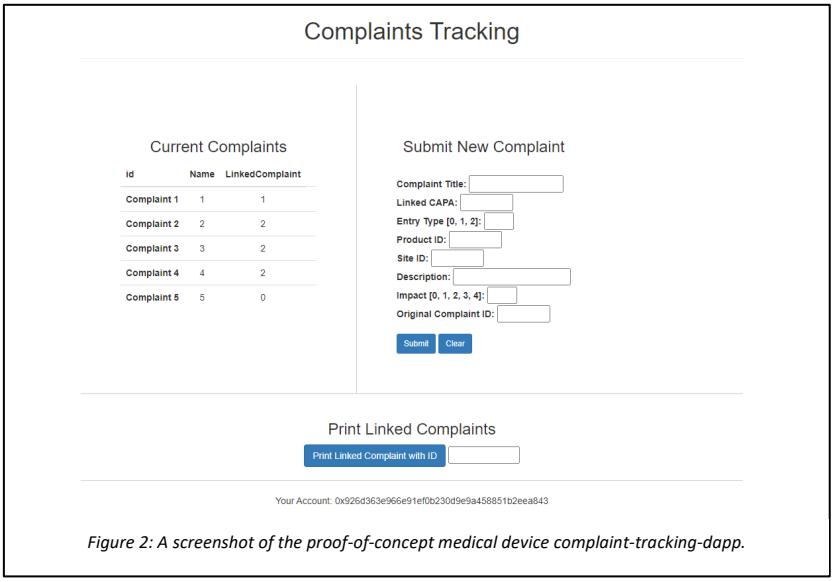

Figure 2: A screenshot of the proof-of-concept medical device complaint-tracking-dapp.

1.6.1.2 Web3 Interface

The web3 interface connects the backend of the blockchain to the front client-facing end. Both Google Chrome and the MetaMask Google Chrome Browser Extension are used to facilitate connecting the user to the locally hosted blockchain. The actual code to connect the browser tools to both the local blockchain and the user was written using Web3.js which is a JavaScript library for building on Ethereum.

[7] Solidity Language

1.6.1.3 Client Facing UI

The client side of the dApp is what the user interacts with. It is composed of the user themselves and the front-end of the dApp which was written using HTML5 and CSS. The result to the user is shown in Figure 2.

This proof-of-concept dApp has been built with five pre-existing block entries on which the functionality can be easily tested. When the dApp is launched, the contract will automatically be populated with these five example complaints.

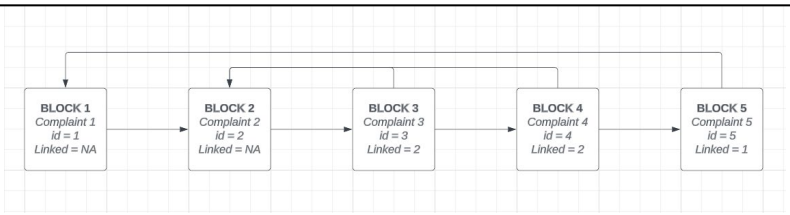

Figure 3: A diagram showing the five example complaints that have been entered into the system and their relation to one another.

1.7 DAPP USAGE

1.7.1.1 Current Complaints

On the left-hand side, and highlighted in Figure 4, there is a field for the current complaints on the chain, populated by the example complaints, each with their own block. Notice that complaints three and four are linked to complaint two, and complaint five is linked to complaint one.

Current Complaints

id	Name	LinkedComplaint
Complaint 1	1	0
Complaint 2	2	0
Complaint 3	3	2
Complaint 4	4	2
Complaint 5	5	1

Figure 5: A Screen shot of the current complaints as populated at launch of the dApp.

Print Linked Complaints

Print Linked Complaint with ID 2

id	Name	LinkedComplaint
2	Complaint 2	0
3	Complaint 3	2
4	Complaint 4	2

Figure 4: A Screen shot of the results of printing all complaints linked with Complaint 2.

1.7.1.2 Print Linked Complaints

When the dApp is launched, you can request the linked complaints be returned as shown in Figure 5.

1.7.1.3 Submitting New Complaints

On the right-hand side of the screen, also shown in Figure 6, is a form to submit a new complaint. If you enter information and hit submit, the contract will check if the entry is valid - i.e., valid entry type, impact type, and linked complaint (if applicable) - before adding it to the chain. If you enter the following dummy data...

You'll see that your MetaMask extension will prompt you for a transaction confirmation. Click 'Submit' to allow the transaction to complete such as shown in Figure 7.

Submit New Complaint

Complaint Title: Complaint 6
Linked CAPA: 1234
Entry Type [0, 1, 2]: 0
Product ID: 8648
Site ID: 2467
Description: Plasma return valve leak
Impact [0, 1, 2, 3, 4]: 4
Original Complaint ID: 1

Submit Clear

Figure 6: A screenshot of the Submit New Complaint fields with example data entered.

Upon completion the screen will refresh and you'll now see Complaint 6 added to the list of current complaints, with block id 6 and linked complaint 1.

1.7.1.4 Your Address

At the bottom of the screenshot in Figure 2, there is an area which indicates which Ethereum wallet address is currently associated with the dApp.

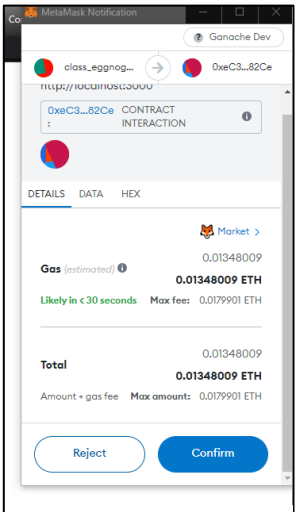

Figure 7: A screenshot showing the prompt from MetaMask to approve a transaction when a new complaint is submitted.

1.8 CONTRACT STRUCTURE

The contract for this medical device proof-of-concept complaint tracking dApp contains contract objects, functions, and testing.

1.8.1 Contract Contents

The contract, named *Complaint*, is made up of a series of complaint entries and containers and objects to store and keep track of those entries.

- *complaintEntry* - A custom struct that models the characteristics of the complaints, where each complaint will be its own block in the chain.
- *entries* - A mapping of arrays of *complaintEntries* to uint keys to keep track of the complaints.
- *entriesIndex* - A mapping of uints to uints to keep track of an entry's position on the block so that it may easily be retrieved.
- *entriesCount* - A numeric count of the number of blocks in the chain.
- Additionally, there is a *submitEvent* event object used to trigger refreshes on the application.

The contract entries are structured as shown in Figure 8 and detailed below. While only the *name*, *id*, and *LinkedComplaint* characteristics are relevant for user interaction, the other characteristics have been created to demonstrate the use case for tracking medical complaints. Additionally, some of the "extra" characteristics are used to ensure entries are valid.

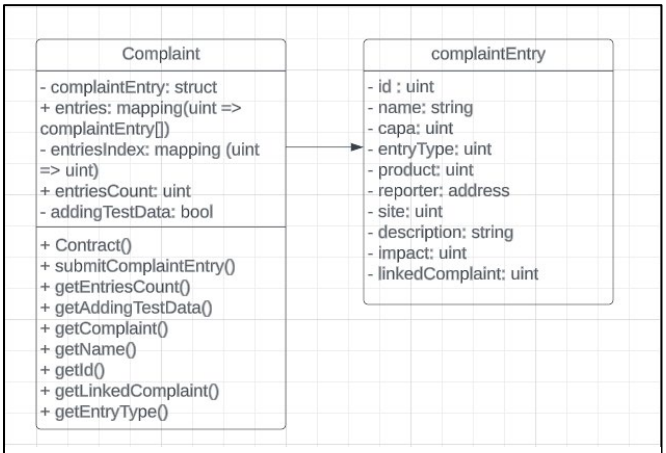

Figure 8: A UML diagram showing the contents of the Contract object, named "Complaint".

- *id* - A unique identifier for the complaint which also matches its block ID on the chain.
- *name* - The user-given name of the complaint.
- *capa* - A numeric identifier to link a complaint to a corrective action plan (capa).
- *entryType* - A numeric enumeration that indicates the type of entry, where 0 = complaint, 1 = addition, 2 = closure.
- *product* - A numeric identifier to map a complaint to a product sku.
- *reporter* - The Ethereum address used to submit the complaint.
- *site* - A numeric identifier to identify the customer site for the complaint.
- *description* - A short description of the complaint.
- *impact* - A numeric enumeration that indicates the impact of the complaint where 0 = observation, 1 = low, 2 = moderate, 3 = high, 4 = SAFETY.
- *linkedComplaint* - A numeric identifier which can be used to link multiple complaints together, 0 indicates no link.

1.8.2 Functions

The contract functions are shown in Figure 8 above. They are used to instantiate the contract, populate dummy data, add complaints to the chain, and retrieve information.

- *Complaint()* - This is the constructor which creates the contract, it also populates the contract with five sample complaints without emitting any events that would cause the application to refresh.

- *submitComplaintEntry()* - Takes in all parameters of a complaint with the exception of id and reporter to submit a new complaint. The id will be given by the contract and will be the number of the next available block. The reporter address will be the Ethereum address associated with the dApp instance that is submitting the complaint.
- *getEntriesCount()* - Returns the number of blocks on the chain.
- *getAddingTestData()* - Returns the value of the boolean *addingTestData* which is used to prevent refresh-triggering events when populating the contract with test data.
- *getComplaint()* - Given a block id (i.e. complaint id) it returns a *complaintEntry* object.
- *getName()* - Given a block id (i.e. complaint id) it returns the name associated with that complaint entry.
- *getId()* - Given a block id (i.e. complaint id) it returns the id associated with that complaint entry.
- *getLinkedComplaint()* - Given a block id (i.e. complaint id) it returns the id of a linked complaint for that complaint entry if it exists.
- *getEntryType()* - Given a block id (i.e. complaint id) it returns the entry type of the complaint entry.

1.8.3 Testing

Contract unit testing was done in JavaScript using the Mocha testing framework. Unit testing of Ethereum contract code is critical to ensure proper functionality of the contract prior to deployment, as it cannot easily be changed post-deployment without migration, which, for large networks can be logistically complex.

A test for all functions in the contract is in the test subdirectory of the dApp. It uses both the existing example data as well as some new data. Recall the original example data populates when you run the contract. Ten different test cases were developed to ensure proper contract functionality, and include the following:

1. Correct Example Data is Generated
2. Correct parameters for test data
3. Emit events don't occur until after test data populates
4. A new complaint can be submitted
5. A valid entry type is required to submit a complaint
6. A valid impact type is required to submit a complaint
7. A valid linked complaint is required to submit a complaint
8. A complaint name can be retrieved given an index
9. A complaint id can be retrieved given an index
10. A complaint's linked ID can be retrieved given an index

2 FUTURE WORK

Future work items have been gathered and published on the project's public-facing project board which can be found at https://github.com/users/siegfrkn/projects/2/views/1.

While over 15 opportunities for improvement have been identified and logged throughout the initial development, they all fall into the following primary categories:

- Deployment to a non-localized Ethereum network
- More robust testing
- Validation of input data against external servers
- Improved linking / lookup functionality

3 CONCLUSION

This dApp serves as a simple proof-of-concept for an Ethereum-based decentralized application for medical device complaint reporting. Given the inherent properties of immutability and distribution of the ledger, this dApp can easily meet both the requirements as outlined by the FDA in 21 CFR 820.198, as well as the common business needs of medical device companies implementing such a system. Future work needs to be done to scale this proof-of-concept to an enterprise-level solution on a non-local blockchain.

4 RESOURCES

1. Blockchain Immutability – Why Does it Matter?
2. Common Medical Device Complaint Problems
3. Best Complaint Management Software
4. Ethereum Whitepaper
5. Ethereum Nodes and Clients
6. Open-Source Wikipedia
7. JPM Coin
8. Solidity Language