VaxDist: A Permissioned Blockchain for the Supply and Distribution of Vaccines

Team 23

Shivam Gandhi
shivam.gandhi@kcl.ac.uk

Loknath Tharun
tharun.loknath@kcl.ac.uk

Thomas Herring
thomas.herring@kcl.ac.uk

Mohit Mukesh Ahuja
mohit_mukesh.ahuja@kcl.ac.uk

Reece Roberts
reece.roberts@kcl.ac.uk

Shreyas Solanki
shreyas.solanki@kcl.ac.uk

April 2021

Contents

1 Executive Summary

Since the release of Nakamoto's Bitcoin in 2009, the development of blockchain and related technologies has accelerated at a staggering pace over the past 10 years. As with all new technologies, thought immediately shifts to alternative implementations and use cases. With the prevalence of the COVID-19 pandemic over the past year, companies and governments have been forced to reconsider the distribution of vaccines at scale, and with disputes arising in the European Union surrounding the supply of vaccines from Astra Zeneca in the UK, and from Pfizer in India, a method must be found which ensures traceability, security, and accountability of vaccine orders - all of which being features that blockchain can facilitate.

In an attempt to cope with the unprecedented demand and tension, we present a permissioned, blockchain-based system: VaxDist, for the management of supply chains in the distribution of vaccines. We explore the use of blockchain technology in the system, evaluate its success, and discuss its potential for production deployment.

2 Goals

At current, existing systems centralise databases which are fundamentally managed by governments or their subsidiaries. This is seen in US' implementation: The Vaccine Tracking System, VTrckS (CDC, 2020), developed and controlled by the Center for Disease Control and Prevention (CDC).

By nature of the blockchain, the "need to utilize an intra-operable vaccine administration system to ensure individual records are not duplicated and monitor reports of adverse events" (Makvandi et al., 2020) is avoided through its key feature of immutability. As such, we are able to keep all parts of the vaccine supply and distribution chain within one system.

Obviously such an approach raises questions about trust and accountability, particularly to those members of the public who have lost confidence in their governments (as we saw in some countries throughout the COVID-19 pandemic of 2020/21). Is the government trying to help us or are they just providing falsified data to create a façade?

The goals we outline below aim to address these issues, with the wholistic aim of providing a guarantee in times of uncertainty.

1. Design and produce a **proof of concept (PoC)** system with:
   (a) **Client nodes** - Anybody who wishes request a shipment of vaccines (eg: Hospitals, Doctors practices, etc...).
   (b) **Distributor nodes** - Those belonging to vaccine manufacturers, or anybody else who is involved in distribution of vaccines (eg: Pfizer, Astra Zeneca, etc...).
2. Create a **graphical user interface** for both types of nodes.
3. Create a **command line interface** that can be used on machines without graphical output.
4. Design a blockchain which allows network nodes to **join and leave** the network at their own leisure.
5. Develop a protocol which validates shipments and requests via a **consensus model**.
6. **Understand** the potential for use of a blockchain based vaccine supply and distribution at scale, and understand the feasibility of the application in a production environment.

3 Methodology

3.1 Design

3.1.1 Use Case Diagram

The use case diagram below outlines the basic actions that clients (top) and distributors (bottom) may undertake in their use of the VaxDist system. All actions, except viewing node consensus power can be executed via either the command line interface (CLI) or graphical user interface (GUI).

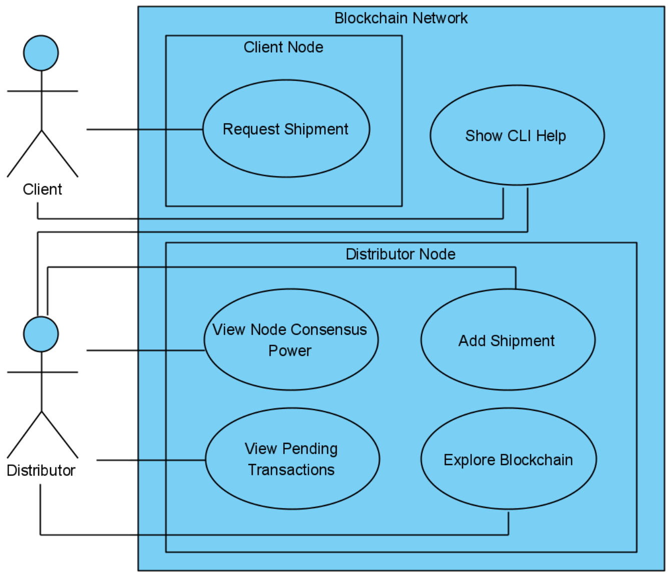


Figure 1: Use Case Diagram

3.1.2 Activity Diagrams

This first activity diagram represents the workflow that occurs on the network once a client (node) places a vaccine order on the network, either via the CLI or GUI.

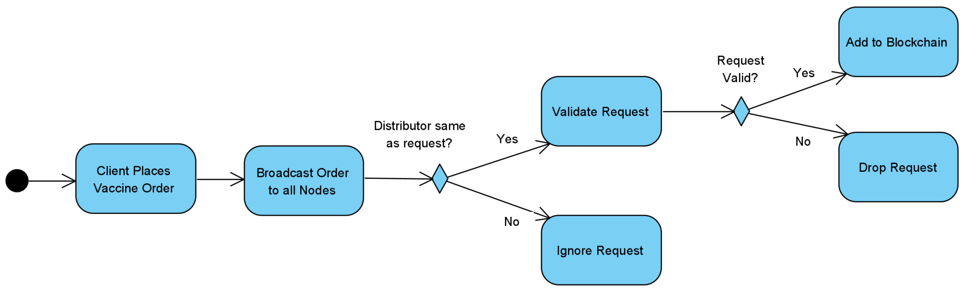

Client Places Vaccine Order
Broadcast Order to all Nodes
Distributor same as request?
Yes
No
Validate Request
Ignore Request
Request Valid?
Yes
No
Add to Blockchain
Drop Request

Figure 2: Request Shipment Activity Diagram

The next diagram shows how the network behaves once a distributor (node) enters shipment information for a pending shipment onto the VaxDist network.

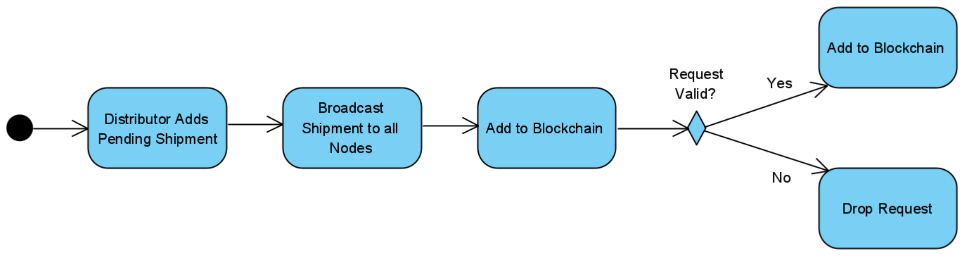


Figure 3: Add Shipment Activity Diagrams

3.2 Supporting Technologies

When deciding on a development environment, it was important to consider the support for blockchain and distributed systems in languages, as well as their available libraries. With this in mind, and based on team experience, we quickly made the decision to implement the blockchain itself in Python 3. A variety of modules were used to assist implementation, the most important of which being RabbitMQ (Rabbit Technologies Ltd., 2007), which provides a message queueing system to allow the network nodes to communicate easily and with little overhead.

The graphical user interface was developed using HTML, Bootstrap, CSS, and JavaScript. An API was developed using Python Flask (Ronacher, 2010) to facilitate interaction with the network nodes. Whilst a frontend web framework such as React may have produced a better UI, we avoided this approach as UI design is out of the scope of the project, and it would likely introduce unnecessary complexities.

3.3 Storing the Blockchain

Each node has its own version of blockchain which is stored when the node goes offline, so as to retrieve it when node comes back online. Python's object serialization library named 'pickle' was used for this purpose. Pickle produces two files:

1. blockchain.pickle - The node's instance of the blockchain (all blocks, their information, and transactions).
2. pending_transactions.pickle - The pending transactions for the current node.

3.4 Block Generation

Each node generates a new block independently to aggregate its pending transactions, and then broadcasts it over the network for validation. A new block may be created in two cases:

1. The limit is reached for the maximum number of pending transactions allowed (5 in our case).
2. The timeout for addition of pending transactions to a block is reached.

4 Results

The system which the team has implemented fulfils all goals, requirements, and hypotheses mentioned in the goals section. The features of this blockchain have been implemented from scratch, in order to cater to all user requirements that would arise in an efficient vaccine logistics system.

In this section we outline what the project has achieved and present our results.

4.1 Goal 1

Design and produce a proof of concept (PoC) system with client nodes and distributor nodes.

Our proof of concept system (VaxDist) is capable of containing both client and distributor nodes. Client's can request a batch of vaccines, and a distributor node can fulfil the request once they are ready to do so.

The use case diagram in figure 1 illustrates the actions that each of these entities can undertake using our system.

4.2 Goal 2

Create a graphical user interface for both types of nodes.

Client nodes and distributor nodes can access the blockchain through a specifically designed graphical user interface (GUI).

Figures 4-8 (Appendices, GUI sub-section) highlight the layout and operation of the GUI.

4.3 Goal 3

Create a command line interface that can be used on machines without graphical output.

Client nodes and distributor nodes can access the blockchain through a command line interface (CLI).

Figure 9 (Appendices, CLI sub-section) shows the operation of the CLI. The interface is the same for both client and distributor nodes, simply with functionality restricted as required for the permitted node functionality.

4.4 Goal 4

Design a blockchain which allows network nodes to join and leave the network at their own leisure.

VaxDist allows client and distributor nodes to join and leave at will. Client nodes are only allowed to request a batch of vaccines from distributors. Distributor nodes, however, are much more capable. Unlike client nodes, they hold the blockchain (and therefore can explore it and view pending transactions), handle block / transaction validation, and submit vaccine shipments.

For example, the blockchain may have many nodes from the company Pfizer. For a transaction, one of the Pfizer nodes will be selected for distribution of the request for a batch of vaccines (based on the transaction created by a client node). The distributor node selected is allowed to leave the network at any time. In such a case, another distributor node will pick up the request for processing.

4.5 Goal 5

Develop a protocol which validates shipments and requests via a consensus model.

The VaxDist system checks all attributes of every shipment at each point of its supply chain path.

Considering a hypothetical example: A package of 10,000 vaccine shots are to be delivered from a factory in Mumbai, India to a hospital in London, UK. This batch of vaccines are manufactured and distributed by Pfizer and is to be taken to its destination location via Dubai, UAE. Our system will ensure that the details of the package are the same when the package is received in Dubai and when it is received in London.

The important attributes of this package will be number of vaccine shots in the package (10,000), distributor name (Pfizer), origin location (Mumbai), destination location (London), etc... In the case of any mismatch of the attributes, the transaction validation will fail, and thus will not be added to the block. This ensures that the package is exactly as ordered by the client, and it remains the same throughout the supply chain path.

4.6 Goal 6

Understand the potential for use of a blockchain based vaccine supply and distribution at scale, and understand the feasibility of the application in a production environment.

The evaluation section covers this goal in detail.

5 Evaluation

The cardinal aim of this project was to design an access control system for a permissioned blockchain for distribution and supply of COVID-19 vaccines. This section will evaluate the different features of the project and characterise them as strengths and weaknesses.

5.1 Strengths of the project

1. Using a blockchain to drive the vaccine supply chain can be especially beneficial as a blockchain allows us to view and track the date, location, quality, certification, and other relevant information essential to maintaining the supply chain. The **block explorer** implemented in the project allows the users to search for a particular piece of data on the blockchain including all the transactions and transaction histories.

2. The transparent track and trace feature of blockchains ensure that only **authorised vaccines** that have been registered and approved are in circulation. This saves the cost of implementing an expensive tracing systems that might burden low and middle income countries.

3. When a client sends a request for a vaccine, the blockchain automatically identifies the correct distributor and only that particular distributor can validate the request. Features like these **automate** the supply chain considerably.

4. A blockchain increases opportunities for planning between suppliers and customers. This enables efficient inventory management and shortage identification while **decreasing risks** associated with forecasting the requirement of vaccines and stock shortages.

5. Blockchain allows for efficient and **fast market recalls** by accurately identifying unsafe vaccines and discovering the exact point of contamination.

6. Caters to **vaccine specific requirements** e.g. the blockchain can be integrated with temperature sensors to ensure refrigeration.

5.2 Weaknesses of the project

1. Establishing an international ecosystem is critical for the blockchain, but this is difficult due to the high cost of implementation and resource requirements. Implementing blockchain solutions also requires significant modification or even complete replacement of existing systems, making it difficult for companies to transition to a blockchain system.

2. A blockchain represents a total shift away from conventional approaches, international organisations might not agree with placing trust and authority in a decentralised network.

6 Future Lessons

In this section we discuss a number potential future improvements of VaxDist, and features that would've been implemented without time constraints. These features are as follows:

- A future version of the network could contain a complete dashboard that shows all the clients and distributors a detailed flow of the order and supply chain. This would enable stakeholders to prioritize their order requirements, and would also give the clients and distributors transparency with respect to the movement of shipments in real time.
- The dashboard could have advanced predictive statics that help the clients understand how the distributors can meet the requirements of an order on time. Machine Learning techniques could be employed to do this.
- The network could be be also be available for doctors (as well as distributors), so they are informed about vaccine availability and can plan their courses of treatment.
- Easy automation of the process of converting the current paper work and structure of the supply chain using AI, enabling a smooth transition from traditional systems to blockchain.
- The future version could contain a different type of consensus so as to make the network dynamically adaptive for various supply chain requirements.
- A priority based system could be useful in future systems, so that disease hotspots get their vaccines as quickly as possible.

7 Bibliography

[1] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[2] CDC. *COVID-19 Vaccine IT Overview: Vaccination Reporting | CDC*. [Online; accessed 4. Apr. 2021]. 2020. URL: https://www.cdc.gov/vaccines/covid-19/reporting/overview/IT-systems.html.

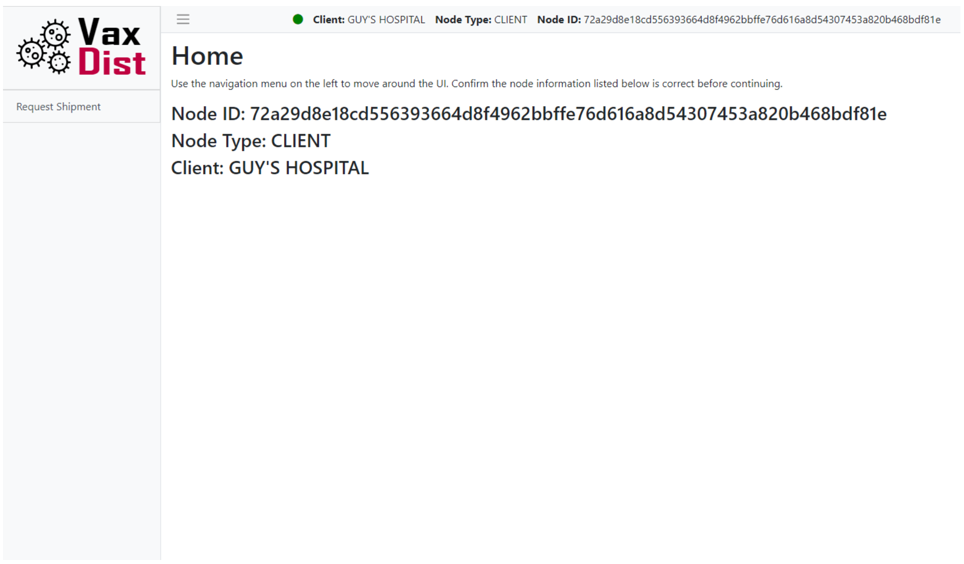
[3] Monear Makvandi et al. *Modeling efficient and equitable distribution of COVID-19 vaccines*. Tech. rep. Oct. 2020. DOI: 10.2172/1718986. URL: https://doi.org/10.2172/1718986.

[4] Rabbit Technologies Ltd. *Messaging that just works — RabbitMQ*. [Online; accessed 4. Apr. 2021]. 2007. URL: https://www.rabbitmq.com.

[5] Armin Ronacher. *Flask*. [Online; accessed 5. Apr. 2021]. 2010. URL: https://palletsprojects.com/p/flask.

7 Appendices

7.1 GUI



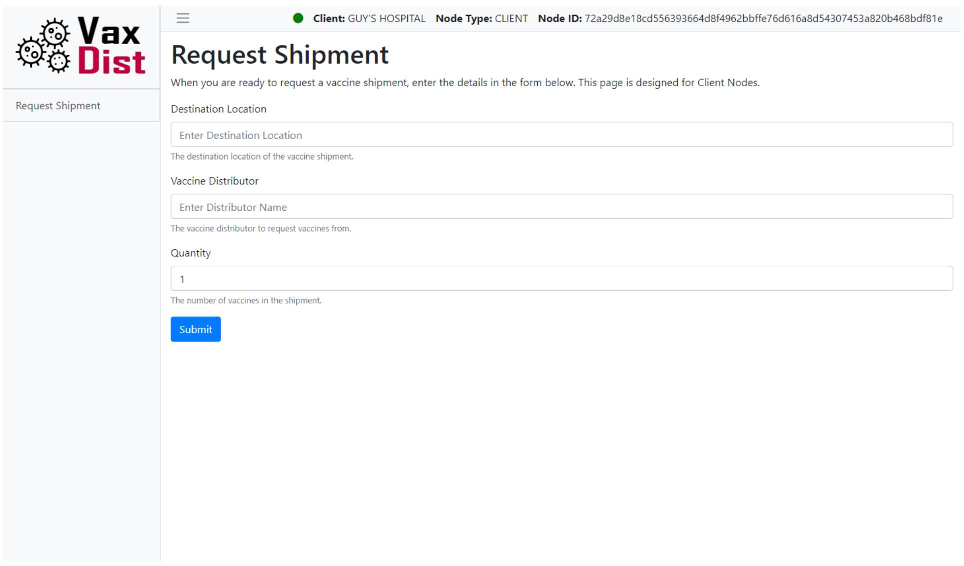
Client: GUY'S HOSPITAL Node Type: CLIENT Node ID: 72a29d8e18cd556393664d8f4962bbffe76d616a8d54307453a820b468bdf81e

Home

Use the navigation menu on the left to move around the UI. Confirm the node information listed below is correct before continuing.

Node ID: 72a29d8e18cd556393664d8f4962bbffe76d616a8d54307453a820b468bdf81e
Node Type: CLIENT
Client: GUY'S HOSPITAL

Figure 4: Landing Page



Client: GUY'S HOSPITAL Node Type: CLIENT Node ID: 72a29d8e18cd556393664d8f4962bbffe76d616a8d54307453a820b468bdf81e

Request Shipment

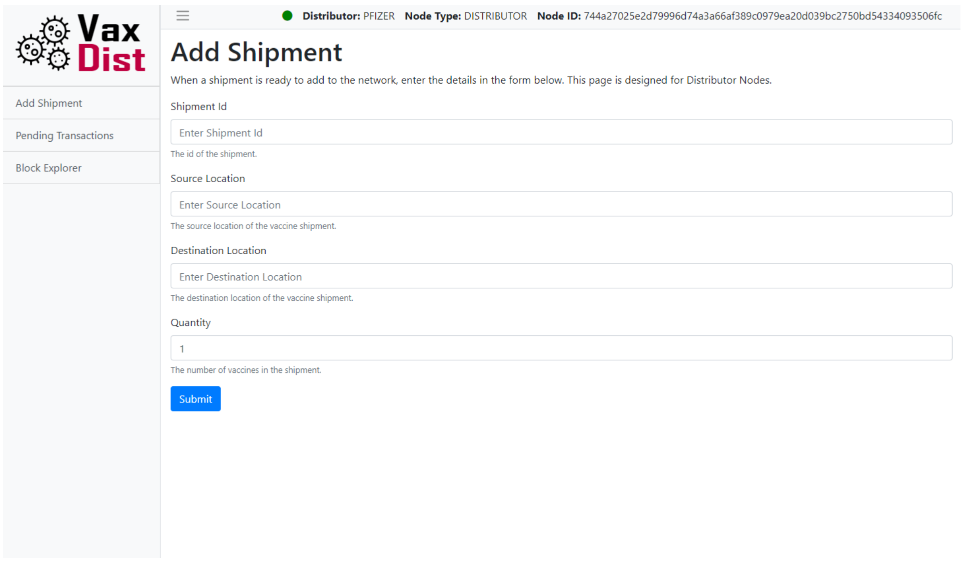
When you are ready to request a vaccine shipment, enter the details in the form below. This page is designed for Client Nodes.

Destination Location: Enter Destination Location
The destination location of the vaccine shipment.

Vaccine Distributor: Enter Distributor Name
The vaccine distributor to request vaccines from.

Quantity: 1
The number of vaccines in the shipment.

Figure 5: Client - Request Shipment



● **Distributor:** PFIZER **Node Type:** DISTRIBUTOR **Node ID:** 744a27025e2d79996d74a3a66af389c0979ea20d039bc2750bd54334093506fc

Add Shipment

When a shipment is ready to add to the network, enter the details in the form below. This page is designed for Distributor Nodes.

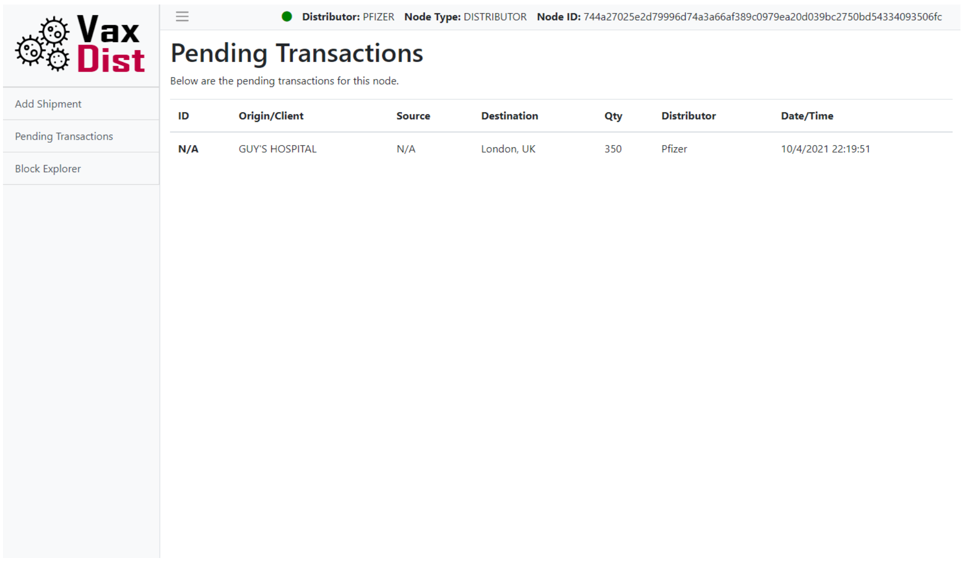
Shipment Id: Enter Shipment Id
The id of the shipment.

Source Location: Enter Source Location
The source location of the vaccine shipment.

Destination Location: Enter Destination Location
The destination location of the vaccine shipment.

Quantity: 1
The number of vaccines in the shipment.

Figure 6: Distributor - Add Shipment



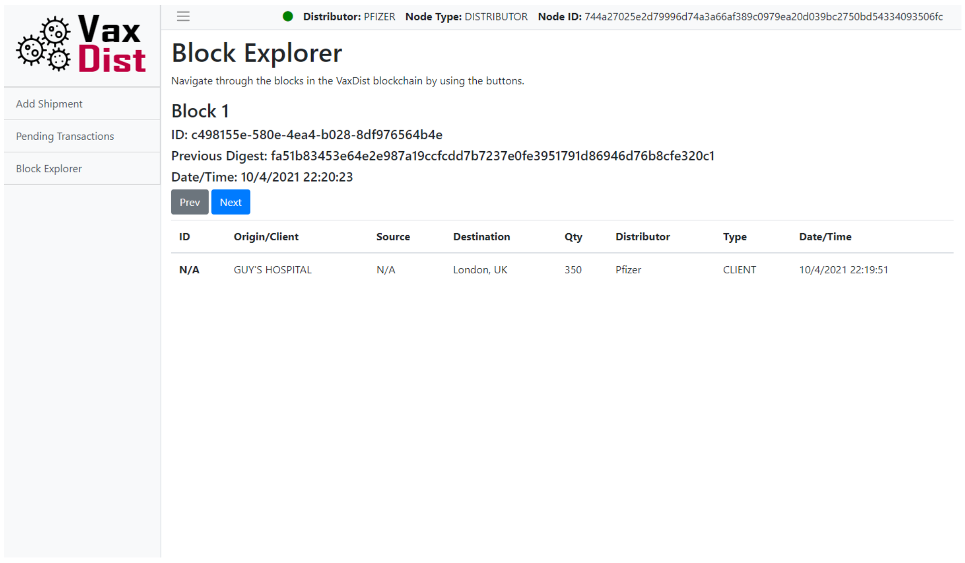
● **Distributor:** PFIZER **Node Type:** DISTRIBUTOR **Node ID:** 744a27025e2d79996d74a3a66af389c0979ea20d039bc2750bd54334093506fc

Pending Transactions

Below are the pending transactions for this node.

ID	Origin/Client	Source	Destination	Qty	Distributor	Date/Time
N/A	GUY'S HOSPITAL	N/A	London, UK	350	Pfizer	10/4/2021 22:19:51

Figure 7: Distributor - View Pending Transactions



Distributor: PFIZER Node Type: DISTRIBUTOR Node ID: 744a27025e2d79996d74a3a66af389c0979ea20d039bc2750bd54334093506fc

Block Explorer

Navigate through the blocks in the VaxDist blockchain by using the buttons.

Block 1

ID: c498155e-580e-4ea4-b028-8df976564b4e

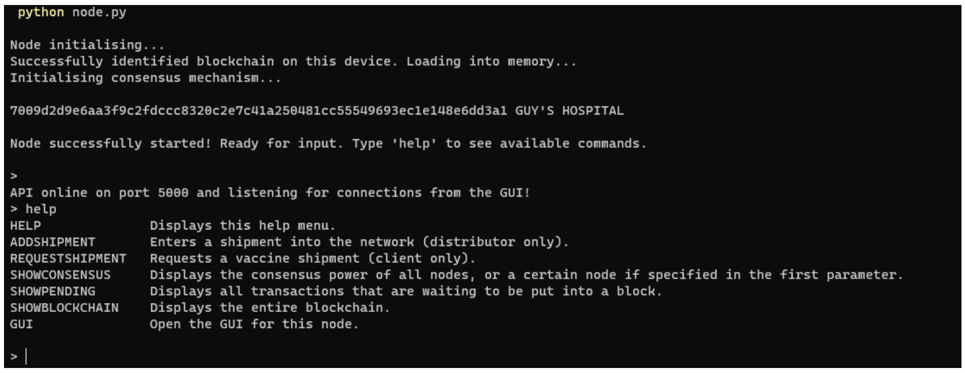
Previous Digest: fa51b83453e64e2e987a19ccfcdd7b7237e0fe3951791d86946d76b8cfe320c1

Date/Time: 10/4/2021 22:20:23

ID	Origin/Client	Source	Destination	Qty	Distributor	Type	Date/Time
N/A	GUY'S HOSPITAL	N/A	London, UK	350	Pfizer	CLIENT	10/4/2021 22:19:51

Figure 8: Distributor - Block Explorer

7.2 CLI



```
 python node.py

Node initialising...
Successfully identified blockchain on this device. Loading into memory...
Initialising consensus mechanism...

7009d2d9e6aa3f9c2fdccc8320c2e7c41a250481cc55549693ec1e148e6dd3a1 GUY'S HOSPITAL

Node successfully started! Ready for input. Type 'help' to see available commands.

>
API online on port 5000 and listening for connections from the GUI!
> help
HELP              Displays this help menu.
ADDSHIPMENT       Enters a shipment into the network (distributor only).
REQUESTSHIPMENT   Requests a vaccine shipment (client only).
SHOWCONSENSUS     Displays the consensus power of all nodes, or a certain node if specified in the first parameter.
SHOWPENDING       Displays all transactions that are waiting to be put into a block.
SHOWBLOCKCHAIN    Displays the entire blockchain.
GUI               Open the GUI for this node.

> |
```

Figure 9: Client/Distributor - Command Line Interface

7.3 Code

```
/
├── node.py
├── blockchain.py
├── block.py
├── communication.py
├── cli.py
├── api.py
├── gui
│   ├── css
│   │   └── style.css
│   ├── js
│   │   ├── add-shipment.js
│   │   ├── api-port.js
│   │   ├── get-block.js
│   │   ├── get-pending-transactions.js
│   │   ├── get-status.js
│   │   └── request-shipment.js
│   ├── add-shipment.html
│   ├── block-explorer.html
│   ├── index.html
│   ├── pending-transactions.html
│   └── request-shipment.html
└── transactions
    ├── client_transaction.py
    └── distributor_transaction.py
```

7.3.1 node.py

```python
from cli import CLI
from communication import Communication
from consensus import Consensus
from api import API
import pickle
import os
import atexit
import signal

from blockchain import Blockchain
import yaml
import hashlib
import random


class Node:
    """
    Singleton class - Can only be one instance of the node class per node.

    Handle all node operations that don't relate to the actual blockchain data structure.
    Networking may be handled here.

    """

    single_instance = None

    def __init__(self):
        self.NODE_ID = None
        self.NODE_NAME = None
        self.NODE_TYPE = None
        self.setup_node()
        # Create the blockchain instance for this node if one doesn't exist
        if not os.path.exists("blockchain.pickle"):
            self.blockchain = Blockchain.instance(node_id=self.NODE_ID, node_name=self.NODE_NAME, node_type=self.NODE_TYPE)
            print("First time booting! Creating genesis block...")
            self.blockchain.create_genesis()
        else:
            print("Successfully identified blockchain on this device. Loading into memory...")
            loaded_blockchain = self.load_blockchain()
            loaded_pending_transactions = self.load_pending_transactions()
            self.blockchain = Blockchain.instance(loaded_blockchain, loaded_pending_transactions, self.NODE_ID, self.NODE_NAME, self.NODE_TYPE)
        API.instance()
        Consensus.instance(self.NODE_NAME)

        self.node_comm = Communication.instance(self.NODE_ID, self.NODE_NAME, self.blockchain, self.NODE_TYPE)
        self.blockchain.set_communication(self.node_comm)
        print(self.NODE_ID, self.NODE_NAME)
        # Send a connection request to other distributor nodes in network
        if self.NODE_TYPE == 'DISTRIBUTOR':
            self.node_comm.notify_connection('connect')

    @staticmethod
    def instance():
        if Node.single_instance == None:
            Node.single_instance = Node()
        return Node.single_instance

    # Recreate the current block so that transactions can continue
    ↪  to be added to it even if
    # the node has stopped midway through the block.
    def __reinstate_current_block(self,
    ↪  loaded_pending_transactions):
        blockchain = self.blockchain.get_blockchain()
        curr_block = blockchain[-1]
        curr_block.set_transactions(loaded_pending_transactions)
        blockchain[-1] = curr_block
        self.blockchain.set_blockchain(blockchain)

    # Save the blockchain to the blockchain.pickle file.
    def dump_blockchain(self):
        print("Saving blockchain state...")
        with open("blockchain.pickle", "wb") as f:
            pickle.dump(self.blockchain.get_blockchain(), f,
            ↪  pickle.HIGHEST_PROTOCOL)

    # Load the blockchain from the blockchain.pickle file.
    def load_blockchain(self):
        with open("blockchain.pickle", "rb") as f:
            return pickle.load(f)

    # Save the pending transactions to the
    ↪  pending_transactions.pickle file.
    def dump_pending_transactions(self):
        print("Saving pending transactions...")
        with open("pending_transactions.pickle", "wb") as f:

            ↪  pickle.dump(self.blockchain.get_pending_transactions(),
            ↪  f, pickle.HIGHEST_PROTOCOL)

    # Load the pending transactions from the
    ↪  pending_transactions.pickle file.
    def load_pending_transactions(self):
        with open("pending_transactions.pickle", "rb") as f:
            loaded_pending_transactions = pickle.load(f)
        return loaded_pending_transactions

    def disconnect(self):
        self.node_comm.notify_connection('disconnect')

    # Setup type of node and its details
    # Store in config.yaml if started first time
    # Else read it from existing config.yaml
    def setup_node(self):
        if os.path.exists("config.yaml"):
            with open("config.yaml", "r") as config:
                loaded_config = yaml.safe_load(config)
                self.NODE_ID = loaded_config["node_id"]
                self.NODE_TYPE = loaded_config["node_type"]
                if self.NODE_TYPE == "DISTRIBUTOR":
                    self.NODE_NAME = loaded_config["distributor"]
                else:
                    self.NODE_NAME = loaded_config["client"]
                config.close()
        else:
            print("No node ID found. Generating one...")
            self.NODE_ID =
            ↪   hashlib.sha256(str(random.getrandbits(256))
            .encode('utf-8')).hexdigest()
            self.NODE_TYPE = input("Are you a client or
            ↪   distributor? (client/distributor) : ").upper()
            while not self.NODE_TYPE in ["CLIENT", "DISTRIBUTOR"]:
                self.NODE_TYPE = input("Are you a client or
                ↪   distributor? (client/distributor) : ").upper()
            self.NODE_NAME = "N/A"
            if self.NODE_TYPE == "DISTRIBUTOR":
                self.NODE_NAME = input(
                    "No distributor found. What distributor do you
                    ↪   belong to? (eg: Pfizer) : ").upper()
                with open("config.yaml", "w") as config:
                    yaml.dump({"node_id": self.NODE_ID,
                    ↪   "distributor": self.NODE_NAME,
                    ↪   "node_type": self.NODE_TYPE},
                              config)
                config.close()

            else:
                self.NODE_NAME = input(
                    "No client found. What is the client's name?
                    ↪    (eg: Guy's Hospital) : ").upper()
                with open("config.yaml", "w") as config:
                    yaml.dump({"node_id": self.NODE_ID, "client":
                    ↪   self.NODE_NAME, "node_type":
                    ↪   self.NODE_TYPE},
                              config)
                    config.close()


if __name__ == "__main__":
    print("\nNode initialising...")
    node = Node.instance()
    atexit.register(node.dump_blockchain)
    atexit.register(node.dump_pending_transactions)
    print("\nNode successfully started! Ready for input. Type
    ↪   'help' to see available commands.")

    command = [""]
    while command[0] not in ["EXIT", "QUIT", "Q"]:
        command = input("\n> ").upper().split(" ")
        CLI(command[0], None or command[1:])

    node.disconnect()
    node.dump_blockchain()
    node.dump_pending_transactions()
    print("\nExiting!")
    os.kill(os.getpid(), signal.SIGTERM)
```

7.3.2 blockchain.py

```python
from communication import Communication
import pickle
import threading
import time
from queue import Queue, Empty

from block import Block
import uuid

from transactions.client_transaction import ClientTransaction
from transactions.distributor_transaction import
↪ DistributorTransaction


class Blockchain:
    """
    Singleton class - Can only be one instance of the blockchain
↪   per node.
    Get the instance via the static instance() method.

    """

    single_instance = None

    comm = None
    node_id = None

    max_pending_transactions = 5
    max_block_wait_time = 60

    candidate_blocks = Queue()
    temp_blocks = dict()
    consensus_results = dict()

    thread_wait = None

    def __init__(self, blockchain, pending_transactions, node_id,
    ↪   node_name, node_type):
        self.blockchain = blockchain
        self.pending_transactions = pending_transactions
        self.node_id = node_id
        self.node_type = node_type
        self.node_name = node_name
        self.start_block_wait_timer()

    def get_node_id(self):
        return self.node_id

    def get_node_name(self):
        return self.node_name

    def get_node_type(self):
        return self.node_type

    def start_block_wait_timer(self):
        self.thread_wait = threading.Thread(target=self.new_block_timer, daemon=True)
        self.thread_wait.start()

    # Display the blockchain nicely when printed.
    def __str__(self):
        blockchain_string = "[\n"
        for b in self.blockchain:
            blockchain_string += str(b)
        blockchain_string += "\n]"
        return blockchain_string

    # STATIC - Fetch and return the single instance of the blockchain, or if it doesn't already exist, instantiate it.
    @staticmethod
    def instance(blockchain=[], pending_transactions=[], node_id=None, node_name=None, node_type=None):
        if Blockchain.single_instance is None:
            Blockchain.single_instance = Blockchain(blockchain, pending_transactions, node_id, node_name, node_type)
        return Blockchain.single_instance

    # Set instance to communicate with network
    def set_communication(self, comm):
        self.comm = comm

    # Fetch the entire blockchain data structure.
    def get_blockchain(self):
        return self.blockchain

    # Set the blockchain data structure.
    def set_blockchain(self, blockchain):
        self.blockchain = blockchain

    # Fetch and return the pending transactions for the current block.
    def get_pending_transactions(self):
        return self.pending_transactions

    # Create the first block in the blockchain.
    def create_genesis(self):
        genesis = Block(
            uuid.uuid4(),  # Block ID
            0,  # block number in blockchain
            "",  # Transactions for this block.
            time.time(),  # Block timestamp.
            "Evening Standard 02/03/2021 Hunt for mutant carrier continues amid row over face masks in schools."
        )
        self.blockchain.append(genesis)
        return genesis

    # Initialise and add a new block to the blockchain.
    def create_block(self):
        print(len(self.pending_transactions))
        block = Block(
            id=str(uuid.uuid4()),  # Block ID.
            block_number=len(self.blockchain),  # Block number
            transactions=self.pending_transactions,  #
            ↪   Transactions for this block.
            timestamp=time.time(),  # Block timestamp.
            previous_digest=Block.get_digest(self.blockchain[-1])
            ↪   # Get the hash digest of the last block.
        )
        self.pending_transactions = []  # Clears pending
        ↪   transactions
        self.temp_blocks[block.id] = block  # Add newly created
        ↪   block to temporary blocks
        self.consensus_results[block.id] = []   # Stores consensus
        ↪   results for this block's validation
        self.comm.broadcast_block(block)    # Broadcast this block
        ↪   over the network for validation
        return block

    # If block is validated based on consensus, add it to
    ↪   blockchain
    def block_validated(self, blockId):
        if blockId not in self.temp_blocks:
            return
        print("\nBlock validated through consensus. Adding",
        ↪   blockId, "to blockchain.\n> ", end="")
        block = self.temp_blocks[blockId]   # Retrieve block from
        ↪   temporary blocks
        self.blockchain.append(block)   # Append block to
        ↪   blockchain
        self.temp_blocks.pop(blockId)   # Remove from temporary
        ↪   blocks
        self.consensus_results.pop(blockId)     # Clear this
        ↪   block's consensus results
        print("\n", str(self), "\n> ", end="")

    # Create a transaction (shipment) and returns it
    def create_distributor_transaction(self, shipment_id,
    ↪   origin_node, src_location, dest_location, qty, type,
                                       distributor):
        transaction = DistributorTransaction(
            shipment_id,
            # self.__generate_shipment_id(distributor),
            origin_node,
            src_location,
            dest_location,
            qty,
```

```python
            distributor,
            type,
            time.time()
        )
        return transaction

    # Create a transaction (request) and returns it
    def create_client_transaction(self, client, dest_location, qty, distributor, type):
        transaction = ClientTransaction(
            # self.__generate_shipment_id(client),
            client,
            dest_location,
            qty,
            distributor,
            type,
            time.time()
        )
        return transaction

    # PRIVATE - Generate a shipment identifier in the format: BlockNum-TransactionInBlockNum/DistributorInitials
    def __generate_distributor_transaction_id(self, distributor):
        block_id = str(len(self.blockchain) - 1)
        transaction_id = str(len(self.pending_transactions))
        distributor_id = "".join([i[0] for i in distributor.split(" ")])

        return block_id + "-" + transaction_id + "/" + distributor_id

    # PRIVATE - Generate a shipment identifier in the format: BlockNum-TransactionInBlockNum/ClientInitials
    def __generate_client_transaction_id(self, client):
        block_id = str(len(self.blockchain) - 1)
        transaction_id = str(len(self.pending_transactions))
        client_id = "".join([i[0] for i in client.split(" ")])

        return block_id + "-" + transaction_id + "/" + client_id

    # Timer thread that creates block with pending transactions on timeout
    def new_block_timer(self):
        while True:
            count = 0
            while count < self.max_block_wait_time and self.thread_wait.is_alive():
                time.sleep(5)
                count += 5
            if len(self.pending_transactions) != 0 and self.thread_wait.is_alive():
                print("\nBlock Timer Expired. Creating block!\n> ", end="")
                self.create_block()
            if self.thread_wait.is_alive() is False:
                self.start_block_wait_timer()
```

7.3.3 block.py

```
import hashlib
import json


class Block:
    """
    Manage the structure of blocks in the blockchain.

    """

    def __init__(self, id, block_number, transactions, timestamp, previous_digest=None):
        self.id = id
        self.block_number = block_number
        self.timestamp = timestamp
        self.transactions = transactions
        self.previous_digest = previous_digest

    # Display the block nicely when printed.
    def __str__(self):
        transactions_string = ""

        transactions_string = "["
        for t in self.transactions:
            transactions_string += str(t)
        transactions_string += "\n            ]"

        block_string = f"""
        {{
            "id": "{self.id}",
            "block_number": "{self.block_number}",
            "timestamp": "{self.timestamp}",
            "transactions": {transactions_string},
            "previous_digest": "{self.previous_digest}"
        }},\n"""
        return block_string

    def get_transactions(self):
        return self.transactions

    def set_transactions(self, transactions):
        self.transactions = transactions

    # STATIC - Get the hash of a given block.
    @staticmethod
    def get_digest(block):
        block_as_json = json.dumps(str(block)).encode()
        digest = hashlib.sha256(block_as_json).hexdigest()
```

```python
        return digest
```

7.3.4 communication.py

```python
import threading
import time
from collections import Set

import jsonpickle
import pika


# Handles the communication between nodes in the blockchain network
class Communication():
    connected_nodes = set()     # Set of connected nodes in the network
    single_instance = None

    my_Lock = threading.Lock()

    def __init__(self, id, node_name, blockchain, node_type):
        self.id = id
        self.node_name = node_name
        self.blockchain = blockchain
        if node_type == 'DISTRIBUTOR':      # Register to listeners only for distributor nodes
            thread_validation = threading.Thread(target=self.register_listeners, args=(), daemon=True)
            thread_validation.start()

    @staticmethod
    def instance(id=None, node_name=None, blockchain=None, node_type=None):
        if Communication.single_instance == None:
            Communication.single_instance = Communication(id, node_name, blockchain, node_type)
        return Communication.single_instance

    # Broadcast all nodes in network
    # status = connect/disconnect
    def notify_connection(self, status):
        connection = pika.BlockingConnection(
            pika.ConnectionParameters(host='localhost'))
        channel = connection.channel()
        channel.exchange_declare(status, exchange_type='fanout')
        channel.basic_publish(exchange=status, routing_key='', body=self.id)
        connection.close()

    # Broadcast all nodes in network a block for validation
    def broadcast_block(self, block):
        connection = pika.BlockingConnection(
            pika.ConnectionParameters(host='localhost'))
        channel = connection.channel()
        channel.exchange_declare('block_validation_request', exchange_type='fanout')
        channel.basic_publish(exchange='block_validation_request', routing_key='',
                              body=jsonpickle.encode(block, unpicklable=True))
        connection.close()

    # Broadcast all nodes in network a transaction for validation
    def broadcast_transaction(self, transction):
        connection = pika.BlockingConnection(
            pika.ConnectionParameters(host='localhost'))
        channel = connection.channel()
        channel.exchange_declare('transaction', exchange_type='fanout')
        channel.basic_publish(exchange='transaction', routing_key='',
                              body=jsonpickle.encode(transction, unpicklable=True))
        connection.close()

    # Broadcast all nodes in network whether a block is validated or not
    def broadcast_block_validation_result(self, result):
        connection = pika.BlockingConnection(
            pika.ConnectionParameters(host='localhost'))
        channel = connection.channel()
        channel.exchange_declare('block_validation_result', exchange_type='fanout')
        channel.basic_publish(exchange='block_validation_result', routing_key='', body=result)
        connection.close()

    # Register for various listeners over the network
    def register_listeners(self):
        connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
        channel = connection.channel()

        # Setup listener for node connection
        channel.exchange_declare('connect', exchange_type='fanout')
        connectionQueue = channel.queue_declare(queue='').method.queue
        channel.queue_bind(connectionQueue, 'connect')
        channel.basic_consume(queue=connectionQueue, on_message_callback=self.node_connected, auto_ack=True)

        # Setup listener for node disconnection
        channel.exchange_declare('disconnect',
        ↪   exchange_type='fanout')
        disconenctionQueue =
        ↪   channel.queue_declare(queue='').method.queue
        channel.queue_bind(disconenctionQueue, 'disconnect')
        channel.basic_consume(queue=disconenctionQueue,
        ↪   on_message_callback=self.node_disconnected,
        ↪   auto_ack=True)

        # Setup listener for block validation approval
        channel.exchange_declare('block_validation_result',
        ↪   exchange_type='fanout')
        validationResultQueue =
        ↪   channel.queue_declare(queue='').method.queue
        channel.queue_bind(validationResultQueue,
        ↪   'block_validation_result')
        channel.basic_consume(queue=validationResultQueue,
        ↪   on_message_callback=self.block_validation_result,
                              auto_ack=True)

        # Setup listener for new transaction in the network
        channel.exchange_declare('transaction',
        ↪   exchange_type='fanout')
        validationResultQueue =
        ↪   channel.queue_declare(queue='').method.queue
        channel.queue_bind(validationResultQueue, 'transaction')
        channel.basic_consume(queue=validationResultQueue,
        ↪   on_message_callback=self.new_transaction,
                              auto_ack=True)

        # Setup listener for block validation request
        channel.exchange_declare('block_validation_request',
        ↪   exchange_type='fanout')
        validationRequestQueue =
        ↪   channel.queue_declare(queue='').method.queue
        channel.queue_bind(validationRequestQueue,
        ↪   'block_validation_request')
        channel.basic_consume(queue=validationRequestQueue,
        ↪   on_message_callback=self.block_validation_request,
                              auto_ack=True)

        channel.start_consuming()

    # Callback when a new node is connected over network
    def node_connected(self, ch, method, properties, body):
        remoteId = str(body).split("'")[1]
        if remoteId != self.id and remoteId not in
        ↪   self.connected_nodes:
            print("\nNode connected: ", remoteId, "\n> ", end="")
            self.connected_nodes.add(remoteId)
            self.notify_connection('connect')

    # Callback when a node is disconnected from network
    def node_disconnected(self, ch, method, properties, body):
```

```python
        remoteId = str(body).split("'")[1]
        if remoteId != self.id and remoteId in self.connected_nodes:
            print("\nNode disconnected: ", remoteId, "\n> ", end="")
            self.connected_nodes.remove(remoteId)

    # Callback when a new pending transaction is received over network
    def new_transaction(self, ch, method, properties, body):
        transaction = jsonpickle.decode(body)
        if transaction not in self.blockchain.pending_transactions:
            if transaction.get_transaction_type() == 'CLIENT':
                if transaction.get_requested_distributor().upper() != self.node_name.upper():
                    return
            print("\nNew request received!\n> ", end="")
            self.blockchain.pending_transactions.append(transaction)

    # Callback on receiving a validation from another node for a block
    def block_validation_result(self, ch, method, properties, body):
        result = str(body).split("'")[1].split(" ")
        node_id = result[0]
        block_id = result[1]
        validationResult = result[2]
        print("\nResult:", result, "\n> ", end="")

        with self.my_Lock:
            if block_id in self.blockchain.temp_blocks:
                block = self.blockchain.temp_blocks[block_id]
                if block.block_number <= self.blockchain.blockchain[-1].block_number:
                    # Discard block since it was late in reaching/requesting consensus
                    # and another block is already added to blockchain
                    # Maybe send approval failure?
                    return
            if validationResult == 'success':
                # Increase consensus count for this block and check if 50% reached
                if block_id in self.blockchain.consensus_results.keys():
                    consensus_results = self.blockchain.consensus_results[block_id]
                    if node_id not in consensus_results:
                        consensus_results.append(node_id)

                    # Reach more than 50% consensus
                    # Change the measure to stake value rather than node count?
                    if len(consensus_results) > ((len(self.connected_nodes) + 1) / 2):
                        # Add approved block to blockchain
                        self.blockchain.block_validated(block_id)
                    else:
                        self.blockchain.consensus_results[block_id] = consensus_results
                else:
                    # Received this block first time. Add to consensus list and wait for 50% approval
                    self.blockchain.consensus_results[block_id] = [node_id]

    # Callback on receiving a validation request for a new block over the network
    def block_validation_request(self, ch, method, properties, body):
        block = jsonpickle.decode(body)

        with self.my_Lock:
            if block.block_number <= self.blockchain.blockchain[-1].block_number:
                # Discard block since it was late in reaching/requesting consensus
                # and another block is already added there
                # Maybe send approval failure?
                return

            print("\nValidate block: ", block, "\n> ", end="")
            time.sleep(10)
            # Received a new block, add it to temp list and wait for 50% consensus
            self.blockchain.temp_blocks[block.id] = block
            if block.id not in self.blockchain.consensus_results.keys():
                self.blockchain.consensus_results[block.id] = []

            self.blockchain.consensus_results[block.id].append(self.id)

        # If block is validated, broadcast it to all nodes
        # Message contains current node's id, block id, approval result
        return self.broadcast_block_validation_result(str(self.id) + " " + str(block.id) + " " + "success")
```

7.3.5 cli.py

```
from communication import Communication
from consensus import Consensus
from blockchain import Blockchain
import webbrowser
import os
from api import API


class CLI:
    def __init__(self, command, params):
        if command == "HELP":
            self.__help()
        elif command == "ADDSHIPMENT":
            self.__addshipment(params)
        elif command == "REQUESTSHIPMENT":
            self.__requestshipment(params)
        elif command == "SHOWCONSENSUS":
            self.__consensus(params)
        elif command == "SHOWPENDING":
            self.__pending_transactions(params)
        elif command == "SHOWBLOCKCHAIN":
            self.__blockchain(params)
        elif command == "GUI":
            self.__gui(params)

        elif command == "":
            pass
        elif not command in ["EXIT", "QUIT", "Q"]:
            print("Invalid input. Type 'help' to see available
             ↪  commands.")

    def __help(self):
        print("HELP              Displays this help menu.")
        print("ADDSHIPMENT       Enters a shipment into the
        ↪  network (distributor only).")
        print("REQUESTSHIPMENT   Requests a vaccine shipment
        ↪  (client only).")
        print("SHOWCONSENSUS     Displays the consensus power of
        ↪  all nodes, or a certain node if specified in the first
        ↪  parameter.")
        print("SHOWPENDING       Displays all transactions that
        ↪  are waiting to be put into a block.")
        print("SHOWBLOCKCHAIN    Displays the entire blockchain.")
        print("GUI               Open the GUI for this node.")

    def __addshipment(self, params):
        if len(params) == 0 or params[0] in ["?", "HELP"]:
            self.__show_command_help(
                usage="addshipment id srclocation destlocation
                ↪  qty",
                description="Enters a shipment into the network."
            )
        elif len(params) == 3:
            shipment_id = params[0]
            src = params[1]
            dest = params[2]
            qty = params[3]

            confirmation = input(
                "Are you sure you want to add this shipment to the
                ↪  blockchain? This action is irreversible
                ↪  without the agreement of all network nodes
                ↪  (y/n): ").upper()

            if confirmation == "Y":
                print("\nAdding to blockchain...")
                blockchain = Blockchain.instance()

                self.node_comm = Communication.instance()

                transaction =
                ↪  blockchain.create_distributor_transaction(shipment_id,
                ↪  blockchain.get_node_id(), src, dest, qty,
                ↪  blockchain.get_node_type(),
                ↪  blockchain.get_node_name())
                self.node_comm.broadcast_transaction(transaction)
                print("Shipment is now pending.\n> ", end="")

    def __requestshipment(self, params):
        if len(params) == 0 or params[0] in ["?", "HELP"]:
            self.__show_command_help(
                usage="requestshipment destlocation qty
                ↪  distributor",
                description="Requests a shipment of vaccines from
                ↪  a given distributor."
            )
        elif len(params) == 3:
            dest = params[0]
            qty = params[1]
            distributor = params[2]

            confirmation = input(
                "Are you sure you want to request this shipment?
                ↪  This action is irreversible without the
                ↪  agreement of all network nodes (y/n):
                ↪  ").upper()

            if confirmation == "Y":
                print("\nAdding to blockchain...")
                blockchain = Blockchain.instance()

                self.node_comm = Communication.instance()

                transaction =
                ↪  blockchain.create_client_transaction(blockchain
                .get_node_name(), dest, qty, distributor,
                ↪  blockchain.get_node_type())

                self.node_comm.broadcast_transaction(transaction)
                print("Request has been received and is now
                ↪  pending.\n> ", end="")

    def __consensus(self, params):
        if len(params) > 0 and params[0] in ["?", "HELP"]:
            self.__show_command_help(
                usage="consensus [node]",
                description="Displays the consensus power of all
                ↪  nodes, or a certain node if specified in the
                ↪  first parameter."
            )
        else:
            consensus_power =
            ↪  Consensus.instance().get_consensus_power()
            if len(params) > 0:
                node = params[0]
                if node in consensus_power:
                    print(node + " : " +
                    ↪  str(consensus_power[node]))
                else:
                    print("Node " + node + " does not exist on the
                    ↪  network. Please try again.")
            else:
                print(consensus_power)

    def __pending_transactions(self, params):
        if len(params) > 0 and params[0] in ["?", "HELP"]:
            self.__show_command_help(
                usage="showpending",
                description="Displays all transactions that are
                ↪  waiting to be put into a block."
            )
        else:
            for transaction in
            ↪  Blockchain.instance().get_pending_transactions():
                print(str(transaction))

    def __blockchain(self, params):
        if len(params) > 0 and params[0] in ["?", "HELP"]:
            self.__show_command_help(
                usage="showblockchain",
                description="Displays the entire blockchain."
            )
        else:
            print(str(Blockchain.instance()))

    # Execute the GUI.
    def __gui(self, params):
        if len(params) > 0 and params[0] in ["?", "HELP"]:
            self.__show_command_help(
                usage="gui",
                description="Open the GUI for this node."
            )
        else:
            webbrowser.open("file:///" + os.getcwd() +
            ↪  "/gui/index.html", new=2)
            with open("gui/js/api-port.js", "w") as f:
                f.write("window.apiPort = " + str(API.instance().get_port()) + ";")

    def __show_command_help(self, usage, description="No description available for this command."):
        print("Usage:\n  " + usage)
        print("Description:\n  " + description)
```

7.3.6 api.py

```
from communication import Communication
from blockchain import Blockchain
from flask import Flask, request, jsonify
from threading import Thread
from flask_cors import CORS, cross_origin
import os
import socket
import webbrowser
import logging
from waitress import serve


class API:
    """
    Singleton class - Can only be one instance of the API per node.
    Get the instance via the static instance() method.

    """

    app = Flask(__name__)

    single_instance = None

    def __init__(self):
        log = logging.getLogger('werkzeug')
        log.setLevel(logging.ERROR)
        cors = CORS(API.app)
        API.app.config['CORS_HEADERS'] = 'Content-Type'
        Thread(target=self.start_server).start()

    @staticmethod
    def instance():
        if API.single_instance == None:
            API.single_instance = API()
        return API.single_instance

    def start_server(self):
        self.port = 5000
        while self.is_port_in_use(self.port):
            self.port += 1
        with open("gui/js/api-port.js", "w") as f:
            f.write("window.apiPort = " + str(self.port) + ";")
```

```python
            f.close()

        print("\nAPI online on port", str(self.port), "and listening for connections from the GUI!\n> ", end="")
        try:
            webbrowser.open("file:///" + os.getcwd() + "/gui/index.html", new=2)
            serve(app=API.app, port=self.port)
        except:
            print("An error occurred when starting the REST API server. Please restart the node and try again. Otherwise, simply use the CLI.")

    def is_port_in_use(self, port):
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            return s.connect_ex(('localhost', port)) == 0

    def get_port(self):
        if API.single_instance:
            return self.port
        return None

    @app.route("/add-shipment", methods=["POST"])
    @cross_origin()
    def add_shipment():
        response = jsonify(success=False, status_code=500)
        if request.method == "POST":
            shipment_id = request.form["shipment-id"]
            src = request.form["src-location"]
            dest = request.form["dest-location"]
            qty = request.form["qty"]

            print("\nAdding to blockchain...")

            blockchain = Blockchain.instance()
            node_comm = Communication.instance()
            transaction = blockchain.create_distributor_transaction(shipment_id, blockchain.get_node_id(), src, dest, qty, blockchain.get_node_type(), blockchain.get_node_name())
            node_comm.broadcast_transaction(transaction)
            print("Shipment is now pending.\n> ", end="")

            response = jsonify(success=True, status_code=200)
        return response

    @app.route("/request-shipment", methods=["POST"])
    @cross_origin()
    def request_shipment():
        response = jsonify(success=False, status_code=500)
        if request.method == "POST":
            dest = request.form["dest-location"]
            qty = request.form["qty"]
            distributor = request.form["distributor"]

            print("\nAdding to blockchain...")

            blockchain = Blockchain.instance()
            node_comm = Communication.instance()
            transaction =
            ↪  blockchain.create_client_transaction(blockchain
            .get_node_name(), dest, qty, distributor,
            ↪  blockchain.get_node_type())
            node_comm.broadcast_transaction(transaction)

            print("Request has been received and is now
            ↪  pending.\n> ", end="")
            response = jsonify(success=True, status_code=200)
        return response

    @app.route("/get-node-info", methods=["GET"])
    @cross_origin()
    def get_distributor():
        blockchain = Blockchain.instance()
        response = jsonify(success=True, status_code=200,
        ↪  distributor_client=blockchain.get_node_name(),
        ↪  node_id=blockchain.get_node_id(),
        ↪  node_type=blockchain.get_node_type())
        return response

    @app.route("/get-pending-transactions", methods=["GET"])
    @cross_origin()
    def get_pending_transactions():
        blockchain = Blockchain.instance()
        pending_transactions =
        ↪  blockchain.get_pending_transactions()
        response = jsonify(success=True, status_code=200,
        ↪  pending_transactions=[str(transaction).strip()[:-1]
        ↪  for transaction in pending_transactions])
        return response

    @app.route("/get-block", methods=["GET"])
    @cross_origin()
    def get_block():
        blockchain = Blockchain.instance().get_blockchain()
        block_index = request.args.get("blockIndex", default=0,
        ↪  type=int)
        if abs(block_index) < len(blockchain):
            block = str(blockchain[block_index - 1]).strip()
            response = jsonify(success=True, status_code=200,
            ↪  block=block[:-1])
        else:
            response = jsonify(success=True, status_code=404)
        return response
```

7.3.7 style.css

```css
#wrapper {
  overflow-x: hidden;
}

#page-content-wrapper {
  min-width: 100vw;
}

#sidebar-wrapper {
  min-height: 100vh;
  margin-left: -15rem;
  -webkit-transition: margin 0.25s ease-out;
  -moz-transition: margin 0.25s ease-out;
  -o-transition: margin 0.25s ease-out;
  transition: margin 0.25s ease-out;
}

#sidebar-wrapper .sidebar-heading {
  padding: 0.875rem 1.25rem;
  font-size: 1.2rem;
}

#sidebar-wrapper .list-group {
  width: 15rem;
}

#wrapper.toggled #sidebar-wrapper {
  margin-left: 0;
}

@media (min-width: 768px) {
  #sidebar-wrapper {
    margin-left: 0;
  }

  #page-content-wrapper {
    min-width: 0;
    width: 100%;
  }

  #wrapper.toggled #sidebar-wrapper {
    margin-left: -15rem;
  }
}

.navbar-toggler-icon:hover {
  cursor: pointer;
}

.alert {
  margin-top: 15px;
}
```

```css

#distributor-client,
#status-indicator,
#node-type,
#node-id {
  padding: 0 15px 0 0;
}

.break-all {
  word-break: break-all;
}

#menu-toggle {
  min-width: 35px;
}

@media (max-width: 1400px) {
  .navbar-nav {
    display: none;
  }
}

.scrollable {
  overflow-x: auto;
}
```

7.3.8 add-shipment.js

```javascript
/**
 * Triggered on submission of the "Add Shipment" form (Distributor
↪  Nodes)
 *
 * Submits the information to the relevant API endpoint.
 */
$("#add-shipment-form").submit(function (event) {
  event.preventDefault();

  $.ajax({
    type: "POST",
    url: "http://localhost:" + window.apiPort + "/add-shipment",
    dataType: "json",
    data: $("#add-shipment-form").serialize(),
    success: function () {
      $("#add-shipment-success").removeClass("d-none");
      $("#add-shipment-failed").addClass("d-none");
      console.log("Request successful!");
    },
    error: function () {
      $("#add-shipment-failed").removeClass("d-none");
      $("#add-shipment-success").addClass("d-none");
      console.log("Request failed. Check that the node is online
      ↪  and try again.");
    },
  });
});
```

7.3.9 api-port.js

```
window.apiPort = 5000;
```

7.3.10 get-block.js

```
window.blockIndex = 0; //0 - most recent, -1 = current block -1,
↪  -2 = current block -2, etc...

/**
 * Render the current block on page load.
 */
$(document).ready(function () {
  getBlock();
});

/**
 * Fetches a block, all of its information, and transactions from
↪  the blockchain.
 * Executed on page load, or on the press of the "Next" and "Prev"
↪  buttons.
 * Renders the transactions and block information on the screen if
↪  in valid range.
 */
function getBlock() {
  $.ajax({
    type: "GET",
    data: { blockIndex: window.blockIndex },
    dataType: "json",
    url: "http://localhost:" + window.apiPort + "/get-block",
    success: function (response) {
      if (response.status_code == 404) {
        console.log("You're at the genesis block!");
        window.blockIndex += 1;
      } else {
        block = JSON.parse(response.block.replaceAll(/},(
        ↪  |\n)*\]/gi, "}]"));
        $("#explorer-block-number").html(block.block_number);
        $("#explorer-block-id").html(block.id);

        ↪  $("#explorer-block-previous-digest").html(block.previous_digest);
        block_timestamp = new Date(Math.trunc(block.timestamp) *
        ↪  1000);
        $("#explorer-block-date-time").html(
          `${block_timestamp.getDate()}/${
            block_timestamp.getMonth() + 1
          }/${block_timestamp.getFullYear()}
          ↪  ${block_timestamp.getHours()}:

↪  ${block_timestamp.getMinutes()}:${block_timestamp.getSeconds()}`
```

```
        );
        if (window.blockIndex == 0) {
          $("#explorer-block-number").append(" (Latest)");
        }

        console.log(block);
        transactions = block.transactions;
        console.log(transactions);
        $("#block-explorer-table-body").html("");

        for (transaction in transactions) {
          transaction = transactions[transaction];
          timestamp = new Date(Math.trunc(transaction.timestamp) *
          ↪  1000);

          if (transaction.transaction_type == "DISTRIBUTOR") {
            midway = Math.round(transaction.origin_node.length /
            ↪  2);
            origin_node = transaction.origin_node.substr(0,
            ↪  midway) + "<wbr>" +
            ↪  transaction.origin_node.substr(midway + 1,
            ↪  transaction.origin_node.length);
          }
          $("#block-explorer-table-body").append(`
              <tr>
                <th scope="row">${transaction.transaction_type ==
↪   "DISTRIBUTOR" ? transaction.shipment_id : "N/A"}</th>
                <td>${transaction.transaction_type ==
↪   "DISTRIBUTOR" ? origin_node : transaction.client}</td>
                <td>${transaction.transaction_type ==
↪   "DISTRIBUTOR" ? transaction.src_location : "N/A"}</td>
                <td>${transaction.dest_location}</td>
                <td>${transaction.qty}</td>
                <td>${transaction.distributor}</td>
                <td>${transaction.transaction_type}</td>
                <td>${timestamp.getDate()}/
                ${timestamp.getMonth() + 1}/
                ${timestamp.getFullYear()}
                ${timestamp.getHours()}:
                ${timestamp.getMinutes()}:
                ${timestamp.getSeconds()}</td>
              </tr>
            `);
        }
        if (transactions.length == 0) {
          $("#block-explorer-table").addClass("d-none");
          //("#no − pending − transactions").removeClass("d − none");
        } else {
          $("#block-explorer-table").removeClass("d-none");
          // ("#no − pending − transactions").addClass("d − none");
        }
      }
    },
    error: function () {
      console.log("Request failed. Check that the node is online and try again.");
    },
  });
}

$("#explorer-prev-btn").click(function () {
  window.blockIndex -= 1;
  getBlock();
});

$("#explorer-next-btn").click(function () {
  if (window.blockIndex < 0) {
    window.blockIndex += 1;
    getBlock();
  }
});
```

7.3.11 get-pending-transactions.js

```
/**
 * Triggered when the user loads the "Pending Transactions" page (Distributor)
 *
 * Retrieves transactions from the appropriate endpoint.
 * Adds all of the transactions to the pending transactions table.
 */
$.ajax({
  type: "GET",
  url: "http://localhost:" + window.apiPort + "/get-pending-transactions",
  success: function (response) {
    pending_transactions = response.pending_transactions;
    for (transaction in pending_transactions) {
      console.log(pending_transactions[transaction]);
      transaction = JSON.parse(pending_transactions[transaction]);
      timestamp = new Date(Math.trunc(transaction.timestamp) * 1000);

      if (transaction.transaction_type == "DISTRIBUTOR") {
        midway = Math.round(transaction.origin_node.length / 2);
        origin_node = transaction.origin_node.substr(0, midway) + "<wbr>" + transaction.origin_node.substr(midway + 1, transaction.origin_node.length);
      }
      $("#pending-transactions-table-body").append(`
          <tr>
            <th scope="row">${transaction.transaction_type == "DISTRIBUTOR" ? transaction.shipment_id : "N/A"}</th>
            <td>${transaction.transaction_type == "DISTRIBUTOR" ? origin_node : transaction.client}</td>
            <td>${transaction.transaction_type == "DISTRIBUTOR" ? transaction.src_location : "N/A"}</td>
            <td>${transaction.dest_location}</td>
            <td>${transaction.qty}</td>
            <td>${transaction.distributor}</td>
            <td>${timestamp.getDate()}/${timestamp.getMonth() + 1}/
            ${timestamp.getFullYear()} ${timestamp.getHours()}:
            ${timestamp.getMinutes()}:${timestamp.getSeconds()}</td>
          </tr>
        `);
    }
    if (pending_transactions.length == 0) {
      $("#pending-transactions-table").addClass("d-none");
      $("#no-pending-transactions").removeClass("d-none");
    } else {
      $("#pending-transactions-table").removeClass("d-none");
      $("#no-pending-transactions").addClass("d-none");
    }
  },
  error: function () {
    console.log("Request failed. Check that the node is online and try again.");
  },
});
```

7.3.12 get-status.js

```
/**
 * Periodically check the status of the node every 10 seconds.
 * Show red icon and appropriate information if the node is offline. Green if all okay.
 * Adapts the UI based on whether the node is a Client or Distributor.
 */
let statusCheck = window.setInterval(
  (function getStatus() {
    $.ajax({
      type: "GET",
      url: "http://localhost:" + window.apiPort + "/get-node-info",
      success: function (response) {
        $("#status-indicator").attr("style", "color: green !important");
        $(".node-status-block").removeClass("d-none");
        $("#node-status-error").addClass("d-none");
        $("#node-id").html(response.node_id);
        $("#distributor-client").html("<strong>" + (response.node_type == "CLIENT" ? "Client: " : "Distributor: ") + "</strong>" + response.distributor_client);
        $("#distributor-client-index").html((response.node_type == "CLIENT" ? "Client: " : "Distributor: ") + response.distributor_client);
        $("#node-id-index").html(response.node_id);
        $("#node-type").html(response.node_type);
        $("#node-type-index").html(response.node_type);
        $("#node-status-index-error").addClass("d-none");
        $("#node-status-index").removeClass("d-none");
        if (response.node_type == "CLIENT") {
          $(".client-nav").removeClass("d-none");
          $(".distributor-nav").addClass("d-none");
        } else {
          $(".client-nav").addClass("d-none");
          $(".distributor-nav").removeClass("d-none");
        }
      },
      error: function () {
        $("#status-indicator").attr("style", "color: red
        ↪  !important");
        $(".node-status-block").addClass("d-none");
        $("#node-status-error").removeClass("d-none");
        $("#node-id").html("");
        $("#node-status-index").addClass("d-none");
        $("#node-status-index-error").removeClass("d-none");
      },
    });
  })(),
  10000
);
```

7.3.13 request-shipment.js

```
/**
 * Triggered on submission of the "Request Shipment" form (Client
↪  Nodes)
 *
 * Submits the information to the relevant API endpoint.
 */
$("#request-shipment-form").submit(function (event) {
  event.preventDefault();

  $.ajax({
    type: "POST",
    url: "http://localhost:" + window.apiPort +
    ↪  "/request-shipment",
    dataType: "json",
    data: $("#request-shipment-form").serialize(),
    success: function () {
      $("#request-shipment-success").removeClass("d-none");
      $("#request-shipment-failed").addClass("d-none");
      console.log("Request successful!");
    },
    error: function () {
      $("#request-shipment-failed").removeClass("d-none");
      $("#request-shipment-success").addClass("d-none");
      console.log("Request failed. Check that the node is online
      ↪  and try again.");
    },
  });
});
```

7.3.14 add-shipment.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,
  ↪  initial-scale=1, shrink-to-fit=no" />

  <title>VaxDist - Add Shipment</title>

  <link rel="stylesheet"
  ↪  href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/
  bootstrap.min.css"
↪  integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PA
  Rn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
    crossorigin="anonymous" />

  <link href="css/style.css" rel="stylesheet" />
  <link rel="shortcut icon" type="image/jpg"
  ↪  href="images/favicon.ico" />
</head>

<body>
  <div class="d-flex" id="wrapper">
    <div class="bg-light border-right" id="sidebar-wrapper">
      <a class="list-group-item list-group-item-action bg-light"
      ↪  href="index.html"><img src="images/logo.svg"></img></a>
      <div class="list-group list-group-flush">
        <div class="distributor-nav d-none">
          <a href="add-shipment.html" class="list-group-item
          ↪  list-group-item-action bg-light">Add Shipment</a>
          <a href="pending-transactions.html"
          ↪  class="list-group-item list-group-item-action
          ↪  bg-light">Pending Transactions</a>
          <a href="block-explorer.html" class="list-group-item
          ↪  list-group-item-action bg-light">Block Explorer</a>
        </div>
        <div class="client-nav d-none"><a
        ↪  href="request-shipment.html" class="list-group-item
        ↪  list-group-item-action bg-light">Request
        ↪  Shipment</a></div>
      </div>
    </div>

    <div id="page-content-wrapper">
      <nav class="navbar navbar-expand-lg navbar-light bg-light
      ↪  border-bottom">
```

```html
      <span class="navbar-toggler-icon border-0" id="menu-toggle"></span>

      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
          <li class="nav-item" id="status-indicator"><i class="fas fa-circle"></i></li>
          <li class="nav-item d-none" id="node-status-error">Node Offline. Check console for errors.</li>
          <li class="nav-item node-status-block"><span id="distributor-client">Querying Node Status...</span></li>
          <li class="nav-item node-status-block d-none"><strong>Node Type:</strong> <span id="node-type"></span></li>
          <li class="nav-item node-status-block d-none"><strong>Node ID:</strong> <span class="break-all" id="node-id"></span></li>
        </ul>
      </div>
    </nav>

    <div class="container-fluid">
      <h1 class="mt-2">Add Shipment</h1>
      <p>When a shipment is ready to add to the network, enter the details in the form below. This page is designed for Distributor Nodes.</p>

      <form id="add-shipment-form">
        <div class="form-group">
          <label for="shipment-id">Shipment Id</label>
          <input required type="text" class="form-control" id="shipment-id" name="shipment-id" aria-describedby="shipment-id" placeholder="Enter Shipment Id" />
          <small class="form-text text-muted">The id of the shipment.</small>
        </div>
        <div class="form-group">
          <label for="src-location">Source Location</label>
          <input required type="text" class="form-control" id="src-location" name="src-location" aria-describedby="src-location" placeholder="Enter Source Location" />
          <small class="form-text text-muted">The source location of the vaccine shipment.</small>
        </div>
        <div class="form-group">
          <label for="dest-location">Destination Location</label>
```

```
            <input required type="text" class="form-control"
            ↪  id="dest-location" name="dest-location"
            ↪  aria-describedby="dest-location"
            ↪  placeholder="Enter Destination Location" />
            <small class="form-text text-muted">The destination
            ↪  location of the vaccine shipment.</small>
          </div>
          <div class="form-group">
            <label for="qty">Quantity</label>
            <input required min="1" value="1" type="number"
            ↪  class="form-control" id="qty" name="qty"
            ↪  aria-describedby="qty" placeholder="Enter
            ↪  Quantity" />
            <small class="form-text text-muted">The number of
            ↪  vaccines in the shipment.</small>
          </div>
          <button type="submit" class="btn btn-primary
          ↪  submit-btn">Submit</button>
        </form>
        <div class="alert alert-success d-none"
        ↪  id="add-shipment-success" role="alert">Successfully
        ↪  added shipment to the blockchain! Click <a
        ↪  href="pending-transactions.html">here</a> to view the
          transaction.</div>
        <div class="alert alert-danger d-none"
        ↪  id="add-shipment-failed" role="alert">Failed to add
        ↪  shipment to blockchain. Check the node is online and
        ↪  try again.</div>
      </div>
    </div>
  </div>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
  ↪  integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
  ↪  crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  js/bootstrap.bundle.min.js"
↪  integrity="sha384-LtrjvnR4Twt/qOuYxE721u19sVFLVSA4hf
  /rRt6PrZTmiPltdZcI7q7PXQBYTKyf"
    crossorigin="anonymous"></script>
  <script src="https://kit.fontawesome.com/4e701a347d.js"
  ↪  crossorigin="anonymous"></script>
  <script src="js/api-port.js"></script>
  <script src="js/add-shipment.js"></script>
  <script src="js/get-status.js"></script>

  <script>
    $("#menu-toggle").click(function (e) {
      e.preventDefault();
      $("#wrapper").toggleClass("toggled");
    });
  </script>
</body>

```

```
</html>
```

7.3.15 block-explorer.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,
   ↪ initial-scale=1, shrink-to-fit=no" />

  <title>VaxDist - Block Explorer</title>

  <link rel="stylesheet"
   ↪ href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  css/bootstrap.min.css"
 ↪ integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PA
  Rn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
    crossorigin="anonymous" />

  <link href="css/style.css" rel="stylesheet" />
  <link rel="shortcut icon" type="image/jpg"
   ↪ href="images/favicon.ico" />
</head>

<body>
  <div class="d-flex" id="wrapper">
    <div class="bg-light border-right" id="sidebar-wrapper">
      <a class="list-group-item list-group-item-action bg-light"
       ↪ href="index.html"><img src="images/logo.svg"></img></a>
      <div class="list-group list-group-flush">
        <div class="distributor-nav d-none">
          <a href="add-shipment.html" class="list-group-item
           ↪ list-group-item-action bg-light">Add Shipment</a>
          <a href="pending-transactions.html"
           ↪ class="list-group-item list-group-item-action
           ↪ bg-light">Pending Transactions</a>
          <a href="block-explorer.html" class="list-group-item
           ↪ list-group-item-action bg-light">Block Explorer</a>
        </div>
        <div class="client-nav d-none"><a
         ↪ href="request-shipment.html" class="list-group-item
         ↪ list-group-item-action bg-light">Request
         ↪ Shipment</a></div>
      </div>
    </div>

    <div id="page-content-wrapper">
      <nav class="navbar navbar-expand-lg navbar-light bg-light
       ↪ border-bottom">
        <span class="navbar-toggler-icon border-0"
         ↪ id="menu-toggle"></span>
```

```html

      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
          <li class="nav-item" id="status-indicator"><i class="fas fa-circle"></i></li>
          <li class="nav-item d-none" id="node-status-error">Node Offline. Check console for errors.</li>
          <li class="nav-item node-status-block"><span id="distributor-client">Querying Node Status...</span></li>
          <li class="nav-item node-status-block d-none"><strong>Node Type:</strong> <span id="node-type"></span></li>
          <li class="nav-item node-status-block d-none"><strong>Node ID:</strong> <span class="break-all" id="node-id"></span></li>
        </ul>
      </div>
    </nav>

    <div class="container-fluid">
      <h1 class="mt-2">Block Explorer</h1>
      <p>Navigate through the blocks in the VaxDist blockchain by using the buttons.</p>
      <h3>Block <span id="explorer-block-number"></span></h3>
      <h5>ID: <span id="explorer-block-id"></span></h5>
      <h5>Previous Digest: <span class="break-all" id="explorer-block-previous-digest"></span></h5>
      <h5>Date/Time: <span id="explorer-block-date-time"></span></h5>

      <div class="scrollable">
        <table class="table d-none mt-2 " id="block-explorer-table">
          <button type="button" class="btn btn-secondary btn mr-1" id="explorer-prev-btn">Prev</button>
          <button type="button" class="btn btn-primary btn" id="explorer-next-btn">Next</button>
          <thead>
            <tr>
              <th scope="col">ID</th>
              <th scope="col">Origin/Client</th>
              <th scope="col">Source</th>
              <th scope="col">Destination</th>
              <th scope="col">Qty</th>
              <th scope="col">Distributor</th>
              <th scope="col">Type</th>
              <th scope="col">Date/Time</th>
            </tr>
          </thead>
          <tbody id="block-explorer-table-body"></tbody>
        </table>
        </div>
      </div>
    </div>
  </div>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
  ↪  integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
  ↪  crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  js/bootstrap.bundle.min.js"
↪   integrity="sha384-LtrjvnR4Twt/qOuYxE721u19sVFLVSA4hf/
  rRt6PrZTmiPltdZcI7q7PXQBYTKyf"
    crossorigin="anonymous"></script>
  <script src="https://kit.fontawesome.com/4e701a347d.js"
  ↪  crossorigin="anonymous"></script>
  <script src="js/api-port.js"></script>
  <script src="js/get-status.js"></script>
  <script src="js/get-block.js"></script>

  <script>
    $("#menu-toggle").click(function (e) {
      e.preventDefault();
      $("#wrapper").toggleClass("toggled");
    });
  </script>
</body>

</html>
```

7.3.16 index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,
  ↪  initial-scale=1, shrink-to-fit=no" />

  <title>VaxDist - Home</title>

  <link rel="stylesheet"
  ↪  href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  css/bootstrap.min.css"
↪   integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PA
  Rn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
    crossorigin="anonymous" />

  <link href="css/style.css" rel="stylesheet" />
  <link rel="shortcut icon" type="image/jpg"
  ↪  href="images/favicon.ico" />
</head>

```

```html
<body>
  <div class="d-flex" id="wrapper">
    <div class="bg-light border-right" id="sidebar-wrapper">
      <a class="list-group-item list-group-item-action bg-light" href="index.html"><img src="images/logo.svg"></img></a>
      <div class="list-group list-group-flush">
        <div class="distributor-nav d-none">
          <a href="add-shipment.html" class="list-group-item list-group-item-action bg-light">Add Shipment</a>
          <a href="pending-transactions.html" class="list-group-item list-group-item-action bg-light">Pending Transactions</a>
          <a href="block-explorer.html" class="list-group-item list-group-item-action bg-light">Block Explorer</a>
        </div>
        <div class="client-nav d-none"><a href="request-shipment.html" class="list-group-item list-group-item-action bg-light">Request Shipment</a></div>
      </div>
    </div>

    <div id="page-content-wrapper">
      <nav class="navbar navbar-expand-lg navbar-light bg-light border-bottom">
        <span class="navbar-toggler-icon border-0" id="menu-toggle"></span><wbr>

        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
            <li class="nav-item" id="status-indicator"><i class="fas fa-circle"></i></li>
            <li class="nav-item d-none" id="node-status-error">Node Offline. Check console for errors.</li>
            <li class="nav-item node-status-block"><span id="distributor-client">Querying Node Status...</span></li>
            <li class="nav-item node-status-block d-none"><strong>Node Type:</strong> <span id="node-type"></span></li>
            <li class="nav-item node-status-block d-none"><strong>Node ID:</strong> <span class="break-all" id="node-id"></span></li>
          </ul>
        </div>
      </nav>

      <div class="container-fluid">
        <h1 class="mt-2">Home</h1>
        <p>Use the navigation menu on the left to move around the UI. Confirm the node information listed below is correct before continuing.</p>
        <h3 class="d-none" id="node-status-index-error">Node
        ↪  Offline! Check console for errors.</h3>
        <span id="node-status-index">
          <h3>Node ID: <span class="break-all"
          ↪  id="node-id-index">Querying...</span></h3>
          <h3>Node Type: <span
          ↪  id="node-type-index">Querying...</span></h3>
          <h3 id="distributor-client-index"></h3>
        </span>
      </div>
    </div>
  </div>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
  ↪  integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
  ↪  crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  js/bootstrap.bundle.min.js"
↪  integrity="sha384-LtrjvnR4Twt/qOuYxE721u19sVFLVSA4hf
  /rRt6PrZTmiPltdZcI7q7PXQBYTKyf"
    crossorigin="anonymous"></script>
  <script src="https://kit.fontawesome.com/4e701a347d.js"
  ↪  crossorigin="anonymous"></script>
  <script src="js/api-port.js"></script>
  <script src="js/get-status.js"></script>

  <script>
    $("#menu-toggle").click(function (e) {
      e.preventDefault();
      $("#wrapper").toggleClass("toggled");
    });
  </script>
</body>

</html>
```

7.3.17 pending-transactions.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,
  ↪  initial-scale=1, shrink-to-fit=no" />

  <title>VaxDist - Pending Transactions</title>

  <link rel="stylesheet"
  ↪  href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  css/bootstrap.min.css"
↪  integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PA
  Rn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
```

```
    crossorigin="anonymous" />

  <link href="css/style.css" rel="stylesheet" />
  <link rel="shortcut icon" type="image/jpg"
  ↪  href="images/favicon.ico" />
</head>

<body>
  <div class="d-flex" id="wrapper">
    <div class="bg-light border-right" id="sidebar-wrapper">
      <a class="list-group-item list-group-item-action bg-light"
      ↪  href="index.html"><img src="images/logo.svg"></img></a>
      <div class="list-group list-group-flush">
        <div class="distributor-nav d-none">
          <a href="add-shipment.html" class="list-group-item
          ↪  list-group-item-action bg-light">Add Shipment</a>
          <a href="pending-transactions.html"
          ↪  class="list-group-item list-group-item-action
          ↪  bg-light">Pending Transactions</a>
          <a href="block-explorer.html" class="list-group-item
          ↪  list-group-item-action bg-light">Block Explorer</a>
        </div>
        <div class="client-nav d-none"><a
        ↪  href="request-shipment.html" class="list-group-item
        ↪  list-group-item-action bg-light">Request
        ↪  Shipment</a></div>
      </div>
    </div>

    <div id="page-content-wrapper">
      <nav class="navbar navbar-expand-lg navbar-light bg-light
      ↪  border-bottom">
        <span class="navbar-toggler-icon border-0"
        ↪  id="menu-toggle"></span>

        <div class="collapse navbar-collapse"
        ↪  id="navbarSupportedContent">
          <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
            <li class="nav-item" id="status-indicator"><i
            ↪  class="fas fa-circle"></i></li>
            <li class="nav-item d-none"
            ↪  id="node-status-error">Node Offline. Check console
            ↪  for errors.</li>
            <li class="nav-item node-status-block"><span
            ↪  id="distributor-client">Querying Node
            ↪  Status...</span></li>
            <li class="nav-item node-status-block
            ↪  d-none"><strong>Node Type:</strong> <span
            ↪  id="node-type"></span></li>
            <li class="nav-item node-status-block
            ↪  d-none"><strong>Node ID:</strong> <span
            ↪  class="break-all" id="node-id"></span></li>
          </ul>
        </div>
      </nav>

      <div class="container-fluid">
        <h1 class="mt-2">Pending Transactions</h1>
        <p>Below are the pending transactions for this node.</p>
        <div class="alert alert-warning d-none"
        ↪   id="no-pending-transactions" role="alert">
          This node currently has no pending transactions. Click
          ↪   <a href="pending-transactions.html">here</a> to
          ↪   refresh.
        </div>
        <div class="scrollable">
          <table class="table d-none"
          ↪   id="pending-transactions-table">
            <thead>
              <tr>
                <th scope="col">ID</th>
                <th scope="col">Origin/Client</th>
                <th scope="col">Source</th>
                <th scope="col">Destination</th>
                <th scope="col">Qty</th>
                <th scope="col">Distributor</th>
                <th scope="col">Date/Time</th>
              </tr>
            </thead>
            <tbody id="pending-transactions-table-body"></tbody>
          </table>
        </div>
      </div>
    </div>
  </div>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
  ↪   integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
  ↪   crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  js/bootstrap.bundle.min.js"
↪   integrity="sha384-LtrjvnR4Twt/qOuYxE721u19sVFLVSA4hf/
  rRt6PrZTmiPltdZcI7q7PXQBYTKyf"
    crossorigin="anonymous"></script>
  <script src="https://kit.fontawesome.com/4e701a347d.js"
  ↪   crossorigin="anonymous"></script>
  <script src="js/api-port.js"></script>
  <script src="js/get-status.js"></script>
  <script src="js/get-pending-transactions.js"></script>

  <script>
    $("#menu-toggle").click(function (e) {
      e.preventDefault();
      $("#wrapper").toggleClass("toggled");
    });
  </script>
</body>

```

```
</html>
```

7.3.18 add-shipment.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,
   ↪ initial-scale=1, shrink-to-fit=no" />

  <title>VaxDist - Request Shipment</title>

  <link rel="stylesheet"
   ↪ href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  css/bootstrap.min.css"
↪ integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PA
  Rn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
    crossorigin="anonymous" />

  <link href="css/style.css" rel="stylesheet" />
  <link rel="shortcut icon" type="image/jpg"
   ↪ href="images/favicon.ico" />
</head>

<body>
  <div class="d-flex" id="wrapper">
    <div class="bg-light border-right" id="sidebar-wrapper">
      <a class="list-group-item list-group-item-action bg-light"
       ↪ href="index.html"><img src="images/logo.svg"></img></a>
      <div class="list-group list-group-flush">
        <div class="distributor-nav d-none">
          <a href="add-shipment.html" class="list-group-item
           ↪ list-group-item-action bg-light">Add Shipment</a>
          <a href="pending-transactions.html"
           ↪ class="list-group-item list-group-item-action
           ↪ bg-light">Pending Transactions</a>
          <a href="block-explorer.html" class="list-group-item
           ↪ list-group-item-action bg-light">Block Explorer</a>
        </div>
        <div class="client-nav d-none"><a
         ↪ href="request-shipment.html" class="list-group-item
         ↪ list-group-item-action bg-light">Request
         ↪ Shipment</a></div>
      </div>
    </div>

    <div id="page-content-wrapper">
      <nav class="navbar navbar-expand-lg navbar-light bg-light
       ↪ border-bottom">
        <span class="navbar-toggler-icon border-0"
         ↪ id="menu-toggle"></span>
```

```html

      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
          <li class="nav-item" id="status-indicator"><i class="fas fa-circle"></i></li>
          <li class="nav-item d-none" id="node-status-error">Node Offline. Check console for errors.</li>
          <li class="nav-item node-status-block"><span id="distributor-client">Querying Node Status...</span></li>
          <li class="nav-item node-status-block d-none"><strong>Node Type:</strong> <span id="node-type"></span></li>
          <li class="nav-item node-status-block d-none"><strong>Node ID:</strong> <span class="break-all" id="node-id"></span></li>
        </ul>
      </div>
    </nav>

    <div class="container-fluid">
      <h1 class="mt-2">Request Shipment</h1>
      <p>When you are ready to request a vaccine shipment, enter the details in the form below. This page is designed for Client Nodes.</p>

      <form id="request-shipment-form">
        <div class="form-group">
          <label for="dest-location">Destination Location</label>
          <input required type="text" class="form-control" id="dest-location" name="dest-location" aria-describedby="dest-location" placeholder="Enter Destination Location" />
          <small class="form-text text-muted">The destination location of the vaccine shipment.</small>
        </div>
        <div class="form-group">
          <label for="distributor">Vaccine Distributor</label>
          <input required type="text" class="form-control" id="distributor" name="distributor" aria-describedby="distributor" placeholder="Enter Distributor Name" />
          <small class="form-text text-muted">The vaccine distributor to request vaccines from.</small>
        </div>
        <div class="form-group">
          <label for="qty">Quantity</label>
          <input required min="1" value="1" type="number" class="form-control" id="qty" name="qty" aria-describedby="qty" placeholder="Enter Quantity" />
```

```
            <small class="form-text text-muted">The number of
            ↪   vaccines in the shipment.</small>
          </div>
          <button type="submit" class="btn btn-primary
          ↪   submit-btn">Submit</button>
        </form>
        <div class="alert alert-success d-none"
        ↪   id="request-shipment-success"
        ↪   role="alert">Successfully added request to the
        ↪   blockchain!</div>
        <div class="alert alert-danger d-none"
        ↪   id="request-shipment-failed" role="alert">Failed to
        ↪   add shipment to blockchain. Check the node is online
        ↪   and try again.</div>
      </div>
    </div>
  </div>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
  ↪   integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
  ↪   crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  js/bootstrap.bundle.min.js"
↪   integrity="sha384-LtrjvnR4Twt/qOuYxE721u19sVFLVSA4hf/
  rRt6PrZTmiPltdZcI7q7PXQBYTKyf"
    crossorigin="anonymous"></script>
  <script src="https://kit.fontawesome.com/4e701a347d.js"
  ↪   crossorigin="anonymous"></script>
  <script src="js/api-port.js"></script>
  <script src="js/get-status.js"></script>
  <script src="js/request-shipment.js"></script>

  <script>
    $("#menu-toggle").click(function (e) {
      e.preventDefault();
      $("#wrapper").toggleClass("toggled");
    });
  </script>
</body>

</html>
```