Final Year Project Technical Documentation

Title Page

- ☐ **Project Title: FarmaTech**
- ☐ **Student Name : Abhijit Mahal & Faizan Ali Shah**
- ☐ **Student ID: 21375106 & 21319001**
- ☐ **Degree Program: Computer Science**
- ☐ **Supervisor Name: Irina Tal**
- ☐ **Institution Name: Dublin City University**
- ☐ **Submission Date: 2nd May 2025**

Table of Contents

1. Abstract

FarmaTech is a blockchain-powered web application designed to improve transparency and security in the pharmaceutical supply chain and provide confidence to the end consumer. Built with Hyperledger Fabric, React.js, MongoDB, and Node.js, it allows key stakeholders drug manufacturers, distributors, regulators, and consumers—to register, verify, and track medicines in real time. The platform uses role-based access control to tailor features to each user type.

Manufacturers can generate unique QR codes, register medicines and supply chain participants, and monitor drug movement. Distributors and regulators can update supply chain data, view analytics, and communicate with manufacturers through an integrated email and notification system. Regulators end the supply chain. End users (consumers) can scan QR codes to verify medicine authenticity, view detailed information, and check if a product is flagged or safe to use. Once a medicine is scanned and claimed, any future scans will reflect its claimed status, preventing misuse. Geolocation further ensures the swift detection and reporting of counterfeit drugs.

2. Introduction

2.1 Background and Motivation

We were interested in how powerful blockchain technology can be, especially beyond just cryptocurrencies like Bitcoin. Blockchain has key features such as no central control, unchangeable records, openness, and strong security, which can help solve many common problems in digital systems.

To showcase the integral features of blockchain that offer a compelling solution to some of the most pressing challenges in digital systems today, we decided to use it in the pharmaceutical industry, where trust and safety is crucial. Fake medicines are a big problem around the world, and many current systems can’t properly track or verify medicines to stop this issue.

Our goal was to build a system that uses blockchain to make sure every medicine record, update, or scan is secure, cannot be changed, and is easy to check. This helps build trust in the system and shows how blockchain can make important systems more reliable and safe, especially in areas where centralised control often leads to errors or abuse.

With this project, we hope to take a small but important step toward a safer and more trustworthy digital future, using the strengths of blockchain technology.

2.2 Objectives:

- **Build a secure blockchain network**
  Use Hyperledger Fabric to create a private blockchain where all medicine records and supply chain actions are safely stored and shared among trusted users.

- **Add QR code tracking and verification**
  Allow manufacturers to create unique QR codes for each medicine so that anyone in the supply chain—including consumers—can scan and check if the medicine is real and safe to use.

- **Set up role-based access**
  Give different access and features to users based on their role (manufacturer, distributor, regulator, or consumer) to keep the system organized and secure.

- **Keep a permanent and unchangeable record**
  Make sure all actions in the supply chain (like registering, moving, or verifying medicines) are saved in a way that cannot be changed, so they can always be checked later if needed.

- **Allow communication between stakeholders**
  Add a simple messaging system so manufacturers, distributors, and regulators can contact each other directly through the platform to share updates or raise concerns.

- **Create an easy-to-use web interface**
  Design simple and clear dashboards using React.js and Node.js so all users can easily manage their tasks, track medicines, and understand the data shown to them.

2.3 Functionality:

- **Manufacturers can:**

    - Register accounts for distributors and regulators
    - Register new medicines and store all details securely on the blockchain
    - Update medicine status across different stages of the supply chain
    - Track real-time movement and historical data of each medicine
    - Generate secure QR codes for every registered medicine
    - View dashboards and monitor your distribution network
    - Communicate directly with your registered distributors and regulators via the platform

- **Distributors and Regulators can:**

    - Update supply chain data as medicines move through different stages
    - Monitor supply chain activity through individual dashboards
    - Flag suspicious or counterfeit medicines

    - Communicate with manufacturers via an in-built messaging and notification system
    - Regulators mark the end of the supply chain.

- **End Users (Consumers) can:**
    - Scan QR codes to verify the authenticity of medicines
    - View detailed information such as origin, batch info, and status
    - See whether a medicine is flagged or safe to use
    - Claim a medicine to prevent re-use or duplicate verification

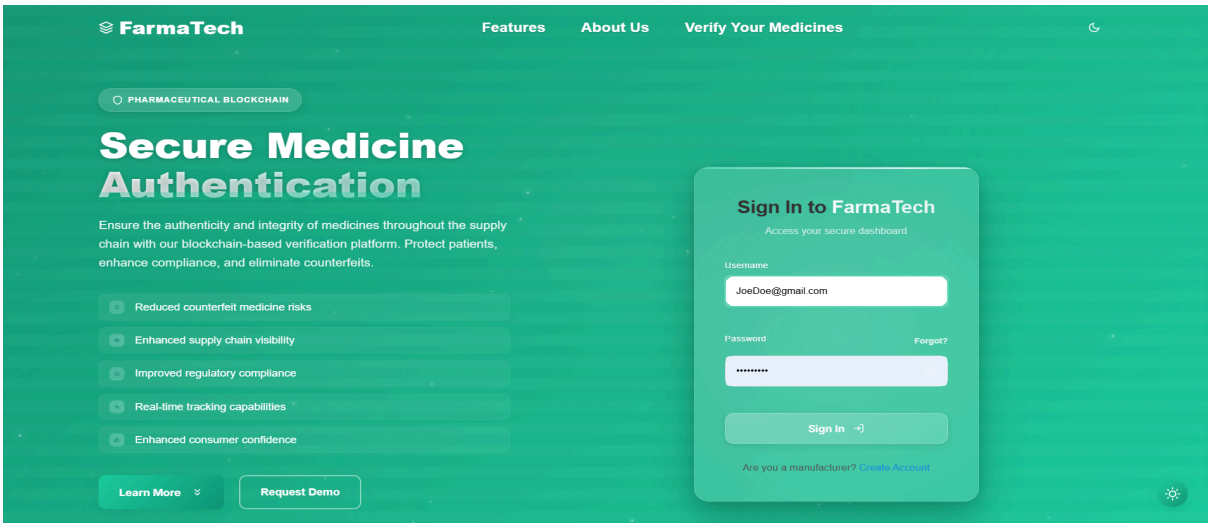
3. Technologies Used

Category	Technology	Purpose
Blockchain Framework	Hyperledger Fabric	Private permissioned blockchain for medicine tracking
Frontend Framework	React	Building responsive, interactive user interfaces
UI Components	MUI (Material-UI), Framer-motion	Styling and animations for a modern, smooth UI
Backend	Node.js	Server-side logic and interaction with Fabric network
Smart Contract SDK	Fabric SDK	Interfacing with blockchain, submitting transactions
Database (Ledger State)	CouchDB	Ledger state database used by Hyperledger Fabric
Database (Off-chain Data)	MongoDB	Storing user profiles & notifications
QR Scanning	ZXing Library	Scanning medicine QR codes for tracking and updates
Email Notifications	SendGrid	Sending emails and notifications
Geolocation Integration	Geolocation API	Capturing and verifying location data
Unit Testing	Mocha + Chai, Jest, bash & Javascript scripting	Writing and running automated test cases

4. System Design

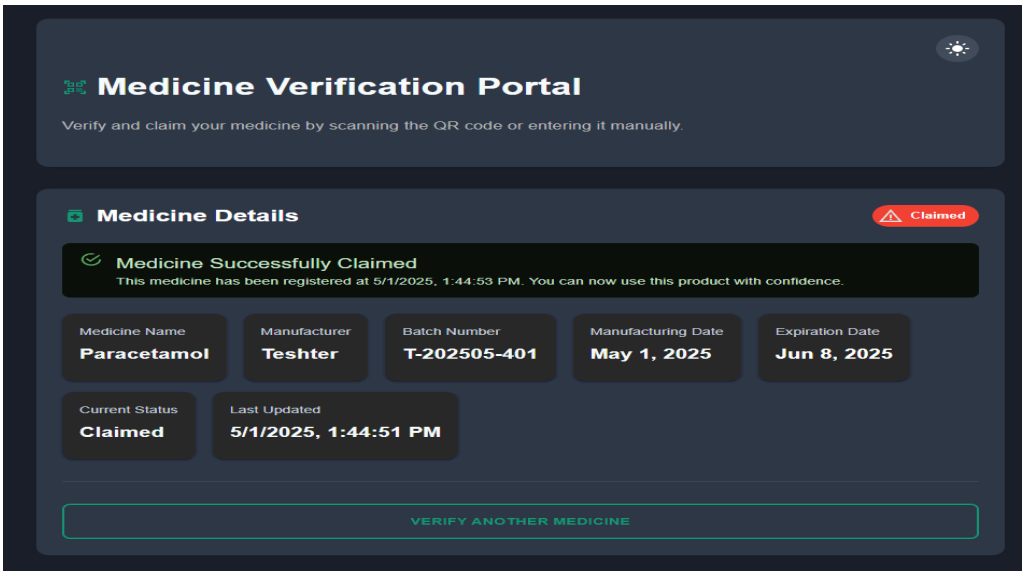
4.1 UI/UX Design

Home Page

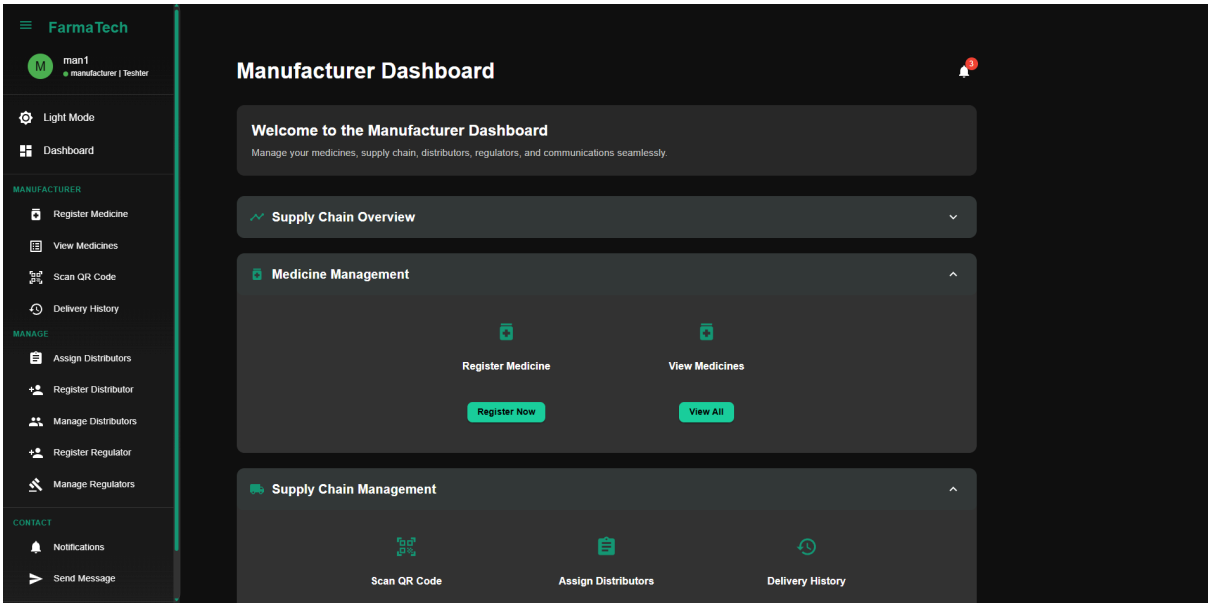


- The design of the page incorporates a basic overview of the project and our achievements and claims.
- The library for design we used was MaterialUI which is used consistently throughout the website. For the home page specifically and its sub-pages, we used the Framer-Motion that allowed us to incorporate simple animation for the visuals we have and the loading screen.
- This page also contains the only end user role page where they can scan and claim their medicine.

Medicine Verification Page

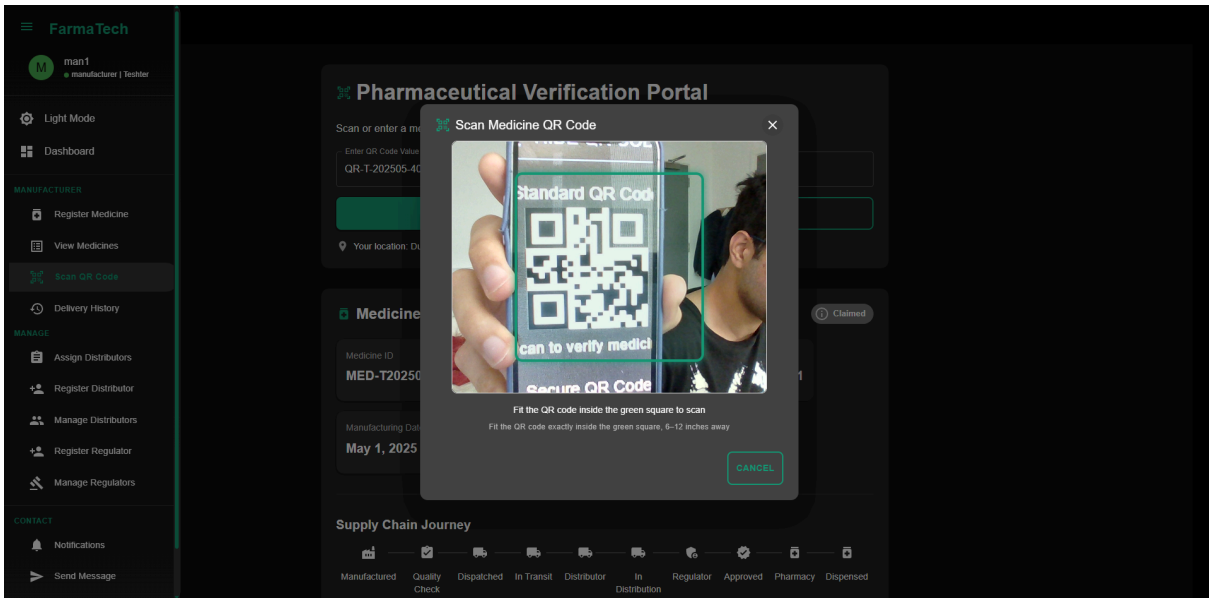


User Dashboards



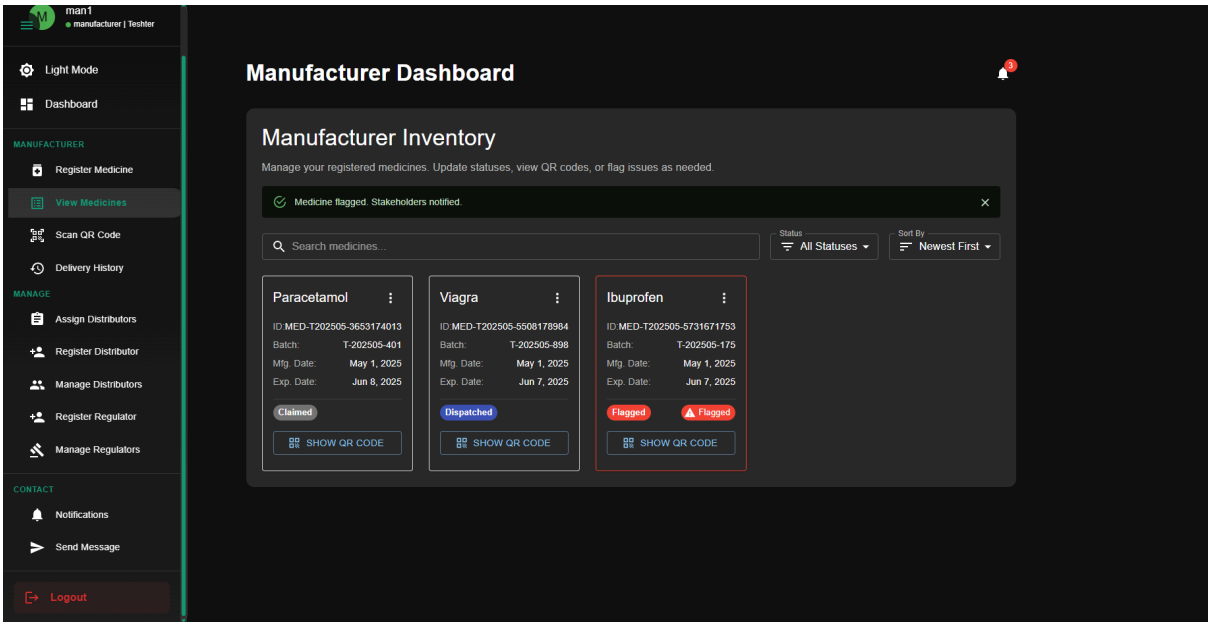
- The dashboard files use the materialUI library for features like the sidebar, that contains all the user specific roles, design of the dropdown menus and buttons.
- Each dashboard UI shares a similar design and also shares common files like viewing the inventory and delivery history to maintain consistency.
- The bell icon on the top right, showcases unread notifications that are generated when a user role contacts another role via the in app email feature.
- Light mode can be toggled from the Light mode button on the sidebar that changes the core color theme of the dashboard.

QR Code Scan



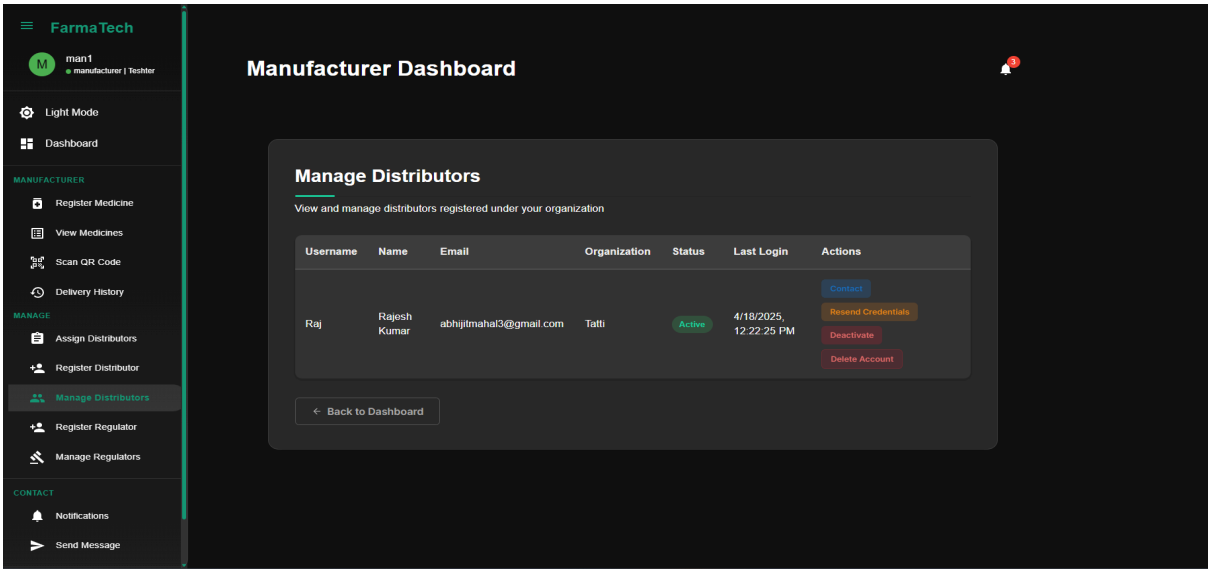
- QR code scanning interface remains the same throughout the dashboards, only differing in terms of what medicine information the user can view.
- The green box indicates the user the optimal area to place the QR code in for the most optimal scan.

User Inventory



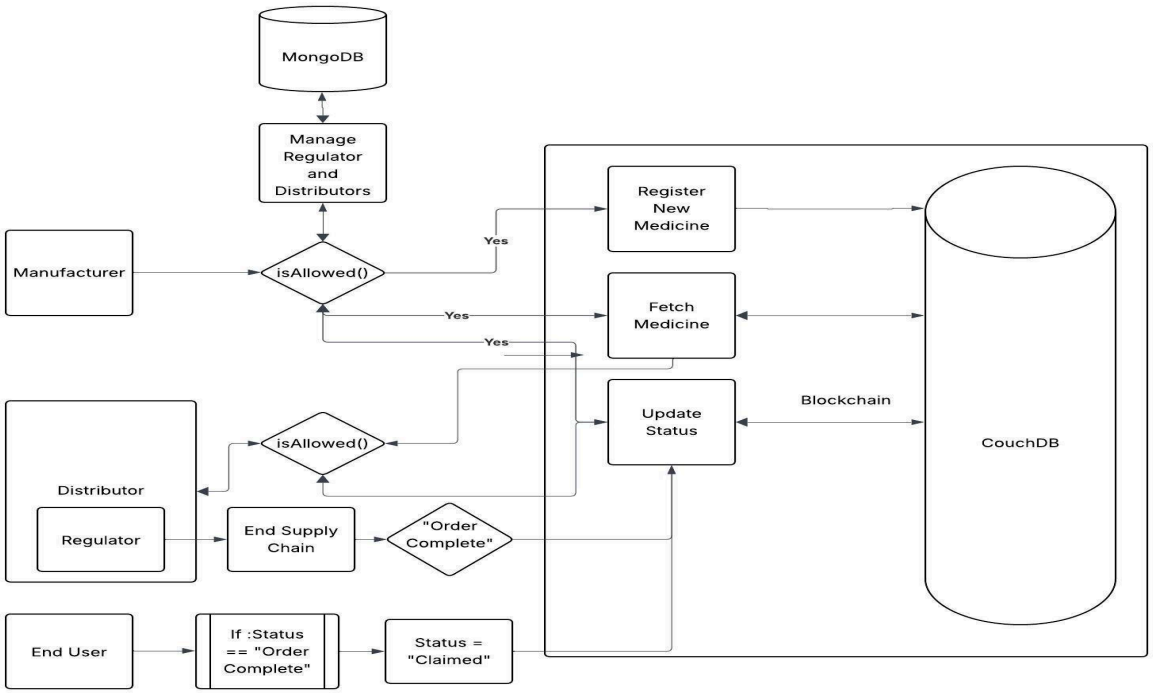
- User inventory also shares a similar design structure where the medicines with their status and basic details are displayed. Medicine can be sorted, filtered and searched for from the top of the inventory.
- For each medicine we also provide, medicine specific functions like flag medicine, update status and contact manufacturer for the specific medicine.

Manage User Dashboard (Manufacturer Specific)



- A manufacturer specific dashboard that allows them to manage their enrolled personnel that are assigned at different points of the supply chain.
- We also provide some core management functionality alongside the basic details of the user information like temporarily deactivate user account, resend credentials, contact and delete.

4.2 Architecture Overview



FarmaTech is architected as a three-tier, blockchain-integrated web application comprising the following layers

Layer	Description
Frontend (React.js)	The UI interacts with users based on their role (Manufacturer, Distributor, Regulator, Consumer).
Backend (Node.js + Express)	The API server facilitates authentication, medicine tracking, QR verification, and blockchain integration.
Blockchain (Hyperledger Fabric)	Secure, permissioned ledger that stores medicine records, supply chain status, and logs.
Database (MongoDB)	Stores user metadata, notifications, and off-chain auxiliary data.

4.2.1 Client Layer

The frontend is built using React.js, delivering responsive, role-specific user interfaces for all stakeholders.

Role-Based Interfaces:

- Manufacturers: Access a comprehensive dashboard to manage medicines, supply chain progress, distributors, regulators, and stakeholder communication.
- Distributors & Regulators: Have dedicated dashboards to scan, track, and update medicine statuses while monitoring their assigned inventory.
- End Users (Consumers): Do not require accounts. They access a public interface to scan QR codes, verify medicine authenticity, and claim products.

Manufacturer Dashboard Features:

- Supply Chain Overview:
  - Medicines with Distributors: % of medicines currently assigned
  - Active Distributors: Count of distributors in the manufacturer's network
  - Avg Distributors per Medicine: Average assignment level
  - Supply Chain Complexity: Qualitative rating (e.g Low)

- Medicine Management: Register new medicines, update records, and generate/view secure QR codes.
- Supply Chain Management: Track live status, movement history, and compliance checkpoints of each medicine
- Distributor Management: Register new distributors and manage access
- Regulator Management: Register and manage assigned regulators for inspection and monitoring
- Communications: Send/receive platform notifications and email-based messages to/from distributors and Regulators

Distributor & Regulator Dashboards:

Distributors and regulators access a tailored dashboard featuring:

- Supply Chain Stats: View summaries of assigned medicines, progress updates, and QR code status
- Scan & Update: Scan medicine QR codes, verify authenticity, and update supply chain status (e.g "In Transit", "Delivered", "Approved")
- Medicine Inventory: View all current and historical medicines associated with their account
- Delivery History: Access timelines and logs of each medicine's path through the supply chain
- Communication Tools: Receive messages from manufacturers and respond directly through the platform
- Notifications: Get notified when flagged, expired, or new medicines are assigned

Authentication & Role-Based Access

Authentication is handled in AuthContext.js using React Context and stored in localStorage

```
const user = userResponse.data;

localStorage.setItem('token', token);
localStorage.setItem('user', JSON.stringify(user));

axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;
setCurrentUser(user);
```

Role-based route protection is implemented using RoleRoute in MedicineRoutes.js:

```
<Routes>
  {/* Dashboard routes */}
  <Route
    path="/manufacturer/*"
    element={
      <RoleRoute
        element={<ManufacturerDashboard />}
        allowedRoles={["manufacturer"]}
      />
    }
  />

  <Route
    path="/distributor/*"
    element={
      <RoleRoute
        element={<DistributorDashboard />}
        allowedRoles={["distributor"]}
      />
    }
  />

  <Route
    path="/regulator/*"
    element={
      <RoleRoute
        element={<RegulatorDashboard />}
        allowedRoles={["regulator"]}
      />
    }
  />
```

Re-Routes the user to the designated dashboard.

Manufacturer Dashboard Features

File: ManufacturerDashboard.js, RegisterNewMedicine.js, QRCodeGenerator.js

Manufacturers can:

- Register medicines(manufacturer specific role):
- RegisterNewMedicine.js

```
  const response = await axios.post(`${API_URL}/medicines`, sanitizedMedicine, {
    headers: { Authorization: `Bearer ${token}` },
  });

  setNewMedicine({
    id: '',
    name: '',
    batchNumber: '',
    manufacturingDate: '',
    expirationDate: '',
    registrationLocation: currentLocation,
    manufacturer: user.organization,
    status: 'Active',
  });

  setSuccessMessage(`Medicine ${response.data.medicine.id} registered successfully!`);
  window.scrollTo({ top: 0, behavior: 'smooth' });
} catch (err) {
  const errorMessage = err.response?.data?.error || 'Failed to register medicine. Please try again.';
  setFormError(errorMessage);
  window.scrollTo({ top: 0, behavior: 'smooth' });
}
```

- View and manage registered medicines (ViewRegisteredMedicines.js)

```
const fetchInventory = async () => {
  setLoading(true);
  setError(null);
  try {
    const response = await axios.get(
      `${API_URL}/medicines/manufacturer/${encodeURIComponent(user.organization)}`,
      {
        headers: { Authorization: `Bearer ${token}` },
      }
    );
    setMedicines(response.data);
    setFilteredMedicines(response.data);
  } catch (err) {
    console.error("Error fetching inventory:", err);
    setError("Failed to fetch inventory. Please try again.");
  } finally {
    setLoading(false);
  }
};
```

- Register distributors (RegisterDistributor.js) and regulators (RegisterRegulator.js)
- Manage distributors (ManageDistributor.js) and manage regulators (ManageRegulator.js)
- Communicate using NotificationForm.js

Distributor & Regulator Dashboards

Files: DistributorDashboard.js, RegulatorDashboard.js, DistributorInventory.js, RegulatorInventory.js.
These roles:

- View inventory using axios.get("/api/medicines/...")

RegulatorInventory.js

```
const fetchInventory = async () => {
  setLoading(true);
  setError(null);

  try {
    const response = await axios.get(`${API_URL}/medicines`, {
      headers: { Authorization: `Bearer ${token}` },
    });

    setMedicines(response.data);
  } catch (err) {
    console.error("Error fetching inventory:", err);
    setError("Failed to fetch inventory. Please try again.");
  } finally {
    setLoading(false);
  }
};
```

DistributorInventory.js

```
const fetchInventory = async () => {
  setLoading(true);
  setError(null);

  try {
    console.log(`Fetching inventory for distributor: ${user.organization}`);
    const response = await axios.get(
      `${API_URL}/medicines/`,

      {
        headers: { Authorization: `Bearer ${token}` },
      }
    );

    console.log(`Received ${response.data.length} medicines from API`);

    if (response.data.length > 0) {
      response.data.forEach((med) => {
        console.log(
          `Medicine ${med.id}: assignedDistributors=${JSON.stringify(
            med.assignedDistributors || []
          )}, includes ${user.organization}=${(
            med.assignedDistributors || []
          ).includes(user.organization)}`
```

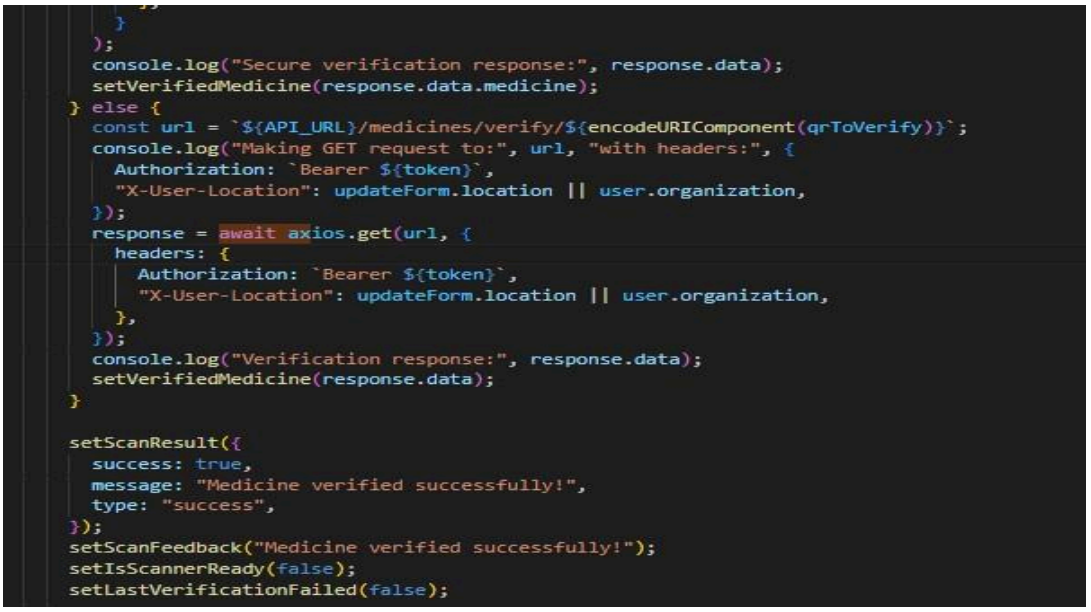
- Track delivery history (DeliveryHistory.js)
- Communicate with manufacturers via Notifications.js

Notification.js

```
const markAsRead = async (id) => {
  try {
    await axios.put(
      `${API_URL}/notifications/${id}/read`,
      {},
      {
        headers: { Authorization: `Bearer ${token}` },
      }
    );

    // Update local state
    setNotifications((prev) =>
      prev.map((notif) =>
        notif._id === id ? { ...notif, isRead: true } : notif
      )
    );
  } catch (error) {
    console.error("Error marking notification as read:", error);
  }
};

const archiveNotification = async (id) => {
  try {
    await axios.put(
      `${API_URL}/notifications/${id}/archive`,
      {},
      {
        headers: { Authorization: `Bearer ${token}` },
      }
    );

    // Update local state by removing the archived notification
    setNotifications((prev) => prev.filter((notif) => notif._id !== id));
```

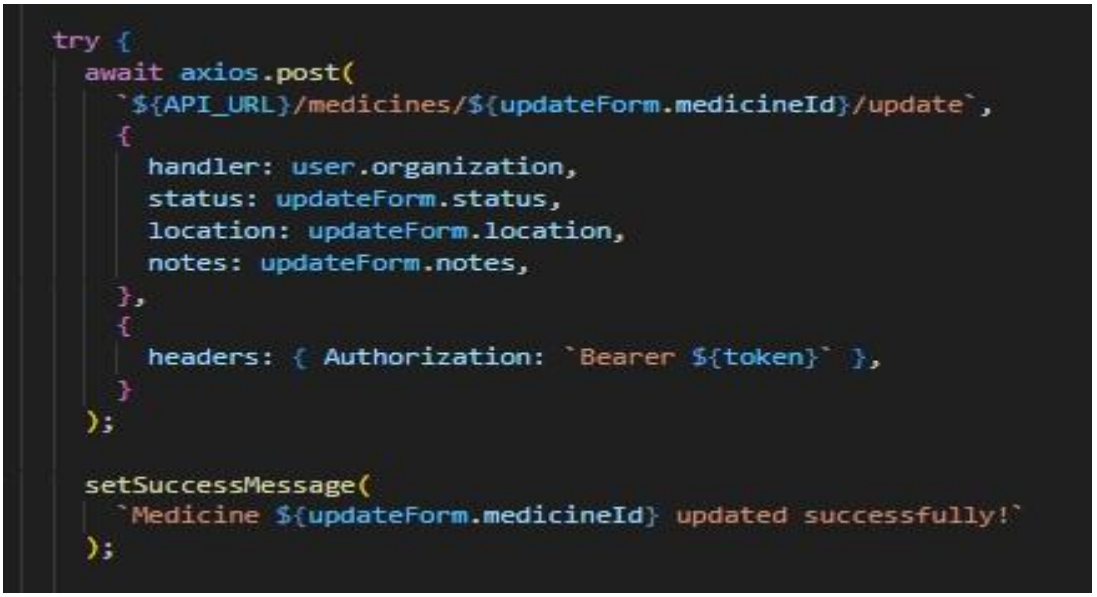
```
const fetchNotifications = async () => {
  setLoading(true);
  try {
    const response = await axios.get(`${API_URL}/notifications`, {
      headers: { Authorization: `Bearer ${token}` },
    });
    setNotifications(response.data);
    setError(null);
  } catch (err) {
    setError("Error loading notifications. Please try again.");
    console.error("Error fetching notifications:", err);
  } finally {
    setLoading(false);
  }
};

useEffect(() => {
  fetchNotifications();
}, [token]);

const handleTabChange = (event, newValue) => {
  setTabValue(newValue);
};
```

- use ScanQRCode.js for QR code scanning:



```
      }
    );
    console.log("Secure verification response:", response.data);
    setVerifiedMedicine(response.data.medicine);
  } else {
    const url = `${API_URL}/medicines/verify/${encodeURIComponent(qrToVerify)}`;
    console.log("Making GET request to:", url, "with headers:", {
      Authorization: `Bearer ${token}`,
      "X-User-Location": updateForm.location || user.organization,
    });
    response = await axios.get(url, {
      headers: {
        Authorization: `Bearer ${token}`,
        "X-User-Location": updateForm.location || user.organization,
      },
    });
    console.log("Verification response:", response.data);
    setVerifiedMedicine(response.data);
  }

  setScanResult({
    success: true,
    message: "Medicine verified successfully!",
    type: "success",
  });
  setScanFeedback("Medicine verified successfully!");
  setIsScannerReady(false);
  setLastVerificationFailed(false);
```

- They can also update supply chain status using ScanQRCode.js:



```
try {
  await axios.post(
    `${API_URL}/medicines/${updateForm.medicineId}/update`,
    {
      handler: user.organization,
      status: updateForm.status,
      location: updateForm.location,
      notes: updateForm.notes,
    },
    {
      headers: { Authorization: `Bearer ${token}` },
    }
  );

  setSuccessMessage(
    `Medicine ${updateForm.medicineId} updated successfully!`
  );
```

Flagging Medicines

Users can flag a suspicious product via FlagMedicine.js:

```
try {

  let reasonText = 'Flagged for issue';
  if (flagForm.reason !== 'other') {
    const selectedReason = flagReasons.find(reason => reason.id === flagForm.reason);
    reasonText = selectedReason ? selectedReason.label : 'Flagged for issue';
  }

  const fullReason = `${reasonText}: ${flagForm.description}`;

  const response = await axios.post(
    `${API_URL}/medicines/${medicine.id}/flag`,
    {
      reason: fullReason,
      location: flagForm.location
    },
    {
      headers: { Authorization: `Bearer ${token}` }
    }
  );
```

Communication System

Implemented using NotificationForm.js and Notifications.js. Users can send messages:

NotificationForm.js

```
const NotificationForm = () => {
  const handleSubmit = async (e) => {
    e.preventDefault();
    setSuccess('');
    setError('');
    if (!validateForm()) return;
    setSending(true);
    try {
      const selectedRecipient = recipients.find(r => r._id === formData.recipientId);
      console.log('Selected Recipient:', {
        id: formData.recipientId,
        role: selectedRecipient.role,
        organization: selectedRecipient.organization,
        username: selectedRecipient.username
      });
      let endpoint;
      const payload = {
        subject: formData.subject,
        message: formData.message
      };
      if (user.role === 'manufacturer') {
        if (selectedRecipient.role === 'distributor') {
          endpoint = `${API_URL}/auth/contact-distributor`;
          payload.distributorId = formData.recipientId;
        } else if (selectedRecipient.role === 'regulator') {
          endpoint = `${API_URL}/auth/contact-regulator`;
          payload.regulatorId = formData.recipientId;
        }
      } else if (user.role === 'distributor' || user.role === 'regulator') {
        endpoint = `${API_URL}/notifications`;
        payload.recipientId = formData.recipientId;
        payload.relatedTo = formData.relatedTo;
        payload.medicineId = formData.medicineId || null;
```

Geolocation Integration

- Used in most components (e.g. scanning, flagging, registration):
- Example: In RegisterNewMedicine.js

```
const detectLocation = async () => {
 setIsDetectingLocation(true);
 if (navigator.geolocation) {
   navigator.geolocation.getCurrentPosition(
     async (position) => {
       const { latitude, longitude } = position.coords;
       try {
         const response = await fetch(
           `https://nominatim.openstreetmap.org/reverse?format=json&lat=${latitude}&lon=${longitude}&zoom=18&addressdetails=1`,
           {
             headers: {
               'Accept-Language': 'en',
               'User-Agent': 'FarmaTech-MedicineApp/1.0',
             },
           }
         );
         if (!response.ok) {
           throw new Error('Failed to get location name');
```

4.2.2. Server Layer

1. **Backend**: A Node.js/Express application serving as middleware between the frontend and Hyperledger Fabric network. It exposes RESTful APIs for medicine registration, flagging, supply chain updates, QR code verification, user management, distributor assignment, and notifications.
    a. The backend is authorised to call the functions of the blockchain if the wallet IDs created in wallet match the id's created by the blockchain.
    b. Execution of each transaction via the chaincode is authorised when the wallet ID matches in both the backend and each container in Docker.
    c. When the blockchain is set up, the certificates created in the process are copied and pasted into the fabric-test-app/config folder.

```
        "connection": {
            "timeout": {
                "peer": {
                    "endorser": "300"
                }
            }
        }
    },
    "organizations": {
        "Org1": {
            "mspid": "Org1MSP",
            "peers": [
                "peer0.org1.example.com"
            ],
            "certificateAuthorities": [
                "ca.org1.example.com"
            ]
        }
    },
    "peers": {
        "peer0.org1.example.com": {
            "url": "grpcs://localhost:7051",
            "tlsCACerts": {
                "pem": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc2gAwIBAgIUGj4ZjosmAzr8CcCilfGF84SozUEwCgYIKoZIzj0EAwIw\ncDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5v
            },
            "grpcOptions": {
                "ssl-target-name-override": "peer0.org1.example.com",
                "hostnameOverride": "peer0.org1.example.com"
            }
        }
    },
    "certificateAuthorities": {
        "ca.org1.example.com": {
            "url": "https://localhost:7054",
            "caName": "ca-org1",
            "tlsCACerts": {
                "pem": ["-----BEGIN CERTIFICATE-----\nMIICJzCCAc2gAwIBAgIUGj4ZjosmAzr8CcCilfGF84SozUEwCgYIKoZIzj0EAwIw\ncDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5v
            },
            "httpOptions": {
                "verify": false
            }
        }
    }
}
```

d. The walled ID for the backend is created by executing the enrollAdmin.js file and registerUser.js file which automatically populates the wallet folder with ID's.

enrollAdmin.js

```javascript
Windsurf: Refactor | Explain | Generate JSDoc | ×
async function main() {
    try {
        const ccpPath = path.resolve(__dirname, 'config', 'connection-org1.json');
        const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'));

        // Create a new CA client for interacting with the CA
        const caInfo = ccp.certificateAuthorities['ca.org1.example.com'];
        const caTLSCACerts = caInfo.tlsCACerts.pem;
        const ca = new FabricCAServices(caInfo.url, { trustedRoots: caTLSCACerts, verify: false }, caInfo.caName);

        // Create a new file system based wallet for managing identities
        const walletPath = path.join(__dirname, 'wallet');
        const wallet = await Wallets.newFileSystemWallet(walletPath);
        console.log(`Wallet path: ${walletPath}`);

        const identity = await wallet.get('admin');
        if (identity) {
            console.log('An identity for the admin user "admin" already exists in the wallet');
            return;
        }

        // Enroll the admin user, and import the new identity into the wallet
        const enrollment = await ca.enroll({ enrollmentID: 'admin', enrollmentSecret: 'adminpw' });
        const x509Identity = {
            credentials: {
                certificate: enrollment.certificate,
                privateKey: enrollment.key.toBytes(),
            },
            mspId: 'Org1MSP',
            type: 'X.509',
        };
        await wallet.put('admin', x509Identity);
        console.log('Successfully enrolled admin user "admin" and imported it into the wallet');

    } catch (error) {
        console.error(`Failed to enroll admin user "admin": ${error}`);
        process.exit(1);
    }
}

main();
```

e. This then gives the backend the authority to communicate and call functions from the blockchain.

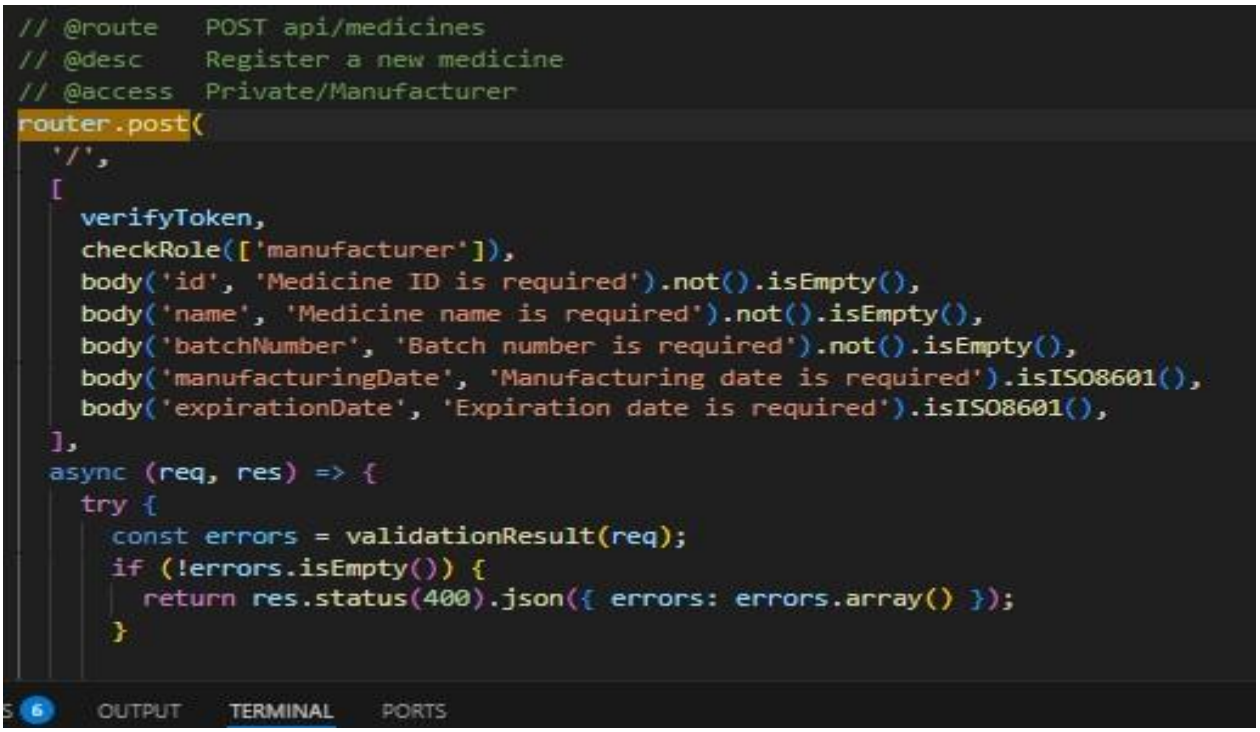
f. After deploying the chaincode successfully we can view all the containers and their role in the blockchain using the "docker -ps" command in the terminal.

```
shahf3@DESKTOP-OPD0N2M:/mnt/c/Users/laptop/Desktop/2025-csc1097-mahala2-shahf3$ docker ps
CONTAINER ID   IMAGE
                                COMMAND                  CREATED        STATUS         PORTS
                                                                NAMES
54f4fc7b9758   dev-peer0.org1.example.com-medicine-contract_1.0-2b8c60104a8fa7bd4ffa311a637dc84ad05689d387482e9c840598405f800c5f-cffc7a3f500452ada4b3ddebf9d7
773099171efe06afc8d57ed879e60526e19d   "docker-entrypoint.s…"   19 hours ago   Up 19 hours
                                                                dev-peer0.org1.example.com-medicine-contract_1.0-2b8c60104a8fa7bd4ffa311a637dc84ad056
89d387482e9c840598405f800c5f
a9bd97fc494a   dev-peer0.org2.example.com-medicine-contract_1.0-2b8c60104a8fa7bd4ffa311a637dc84ad05689d387482e9c840598405f800c5f-c94b7949f6f16f68792392582b52
bb11cd7dbf88443c62d67aeee516715ecbde   "docker-entrypoint.s…"   19 hours ago   Up 19 hours
                                                                dev-peer0.org2.example.com-medicine-contract_1.0-2b8c60104a8fa7bd4ffa311a637dc84ad056
89d387482e9c840598405f800c5f
326f40039cc1   hyperledger/fabric-peer:latest
                                "peer node start"        19 hours ago   Up 19 hours    0.0.0.0:9051->9051/tcp, [::]:9051->9051/tcp, 7051/tcp, 0.0.0.0:9
445->9445/tcp, [::]:9445->9445/tcp                              peer0.org2.example.com
cd89f007764d   hyperledger/fabric-peer:latest
                                "peer node start"        19 hours ago   Up 19 hours    0.0.0.0:7051->7051/tcp, [::]:7051->7051/tcp, 0.0.0.0:9444->9444/
tcp, [::]:9444->9444/tcp                                        peer0.org1.example.com
f1fa78503738   hyperledger/fabric-orderer:latest
                                "orderer"                19 hours ago   Up 19 hours    0.0.0.0:7050->7050/tcp, [::]:7050->7050/tcp, 0.0.0.0:7053->7053/
tcp, [::]:7053->7053/tcp, 0.0.0.0:9443->9443/tcp, [::]:9443->9443/tcp   orderer.example.com
d9a11123ad09   couchdb:3.3.3
                                "tini -- /docker-ent…"   19 hours ago   Up 19 hours    4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp, [::]:7984->5984/tcp
                                                                couchdb1
d33c9bdc1939   couchdb:3.3.3
                                "tini -- /docker-ent…"   19 hours ago   Up 19 hours    4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp, [::]:5984->5984/tcp
                                                                couchdb0
01fb5bd960eb   hyperledger/fabric-ca:latest
                                "sh -c 'fabric-ca-se…"   19 hours ago   Up 19 hours    0.0.0.0:9054->9054/tcp, [::]:9054->9054/tcp, 7054/tcp, 0.0.0.0:1
9054->19054/tcp, [::]:19054->19054/tcp                          ca_orderer
30ece6935dbe   hyperledger/fabric-ca:latest
                                "sh -c 'fabric-ca-se…"   19 hours ago   Up 19 hours    0.0.0.0:7054->7054/tcp, [::]:7054->7054/tcp, 0.0.0.0:17054->1705
4/tcp, [::]:17054->17054/tcp                                    ca_org1
089aad0514bc   hyperledger/fabric-ca:latest
                                "sh -c 'fabric-ca-se…"   19 hours ago   Up 19 hours    0.0.0.0:8054->8054/tcp, [::]:8054->8054/tcp, 7054/tcp, 0.0.0.0:1
8054->18054/tcp, [::]:18054->18054/tcp                          ca_org2
```

- Key Modules:
    - **medicines.js:** Handles medicine-related blockchain transactions (e.g registration, status updates, flagging) and QR code generation/verification.
    - **auth.js:** Manages user authentication (JWT), registration (manufacturers, distributors, regulators), and email notifications via SendGrid.
    - **notifications.js:** Supports in-system notifications and email alerts for communication between roles.
- Fabric SDK: Uses Hyperledger Fabric Node SDK to interact with the blockchain network, invoking chaincode functions like RegisterMedicine, UpdateSupplyChain, and AssignDistributor.
- Authentication: Implements JWT-based authentication for private endpoints, with public endpoints for end-user verification.
- Security: unauthorized scan detection.

Code Snapshot: Express Route (Register Medicine)

From fabric-test-app/routes/medicines.js



Code Snapshot: Express Route (Update Supply Chain)
From fabric-test-app/routes/medicines.js

```
// @route   POST api/medicines/:id/update
// @desc    Update medicine supply chain or compliance status
// @access  Private/Distributor, Manufacturer, Regulator, Enduser
router.post(
  '/:id/update',
  [
    verifyToken,
    checkRole(['distributor', 'manufacturer', 'regulator', 'enduser']),
    body('status', 'Status is required').not().isEmpty(),
    body('location', 'Location is required').not().isEmpty(),
  ],
  async (req, res) => {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
      }

      const { id } = req.params;
      const { status, location, notes } = req.body;
```

Code Snapshot: Express Route (Public QR Code Verification)
From fabric-test-app/routes/medicines.js

```
// @route   GET api/public/verify/:qrCode
// @desc    Verify medicine by QR code (public access)
// @access  Public
router.get('/public/verify/:qrCode', async (req, res) => {
  try {
    const { qrCode } = req.params;
    const location = req.headers['x-user-location'] || 'Unknown';

    if (!qrCode || !qrCode.startsWith('QR-')) {
      return res.status(400).json({ error: 'Invalid QR code format. Must start with "QR-"' });
    }

    const walletPath = path.join(__dirname, "../wallet");
    const wallet = await Wallets.newFileSystemWallet(walletPath);
    const identity = await wallet.get("appUser");
    if (!identity) {
      return res.status(400).json({ error: 'User "appUser" does not exist in the wallet' });
    }
```

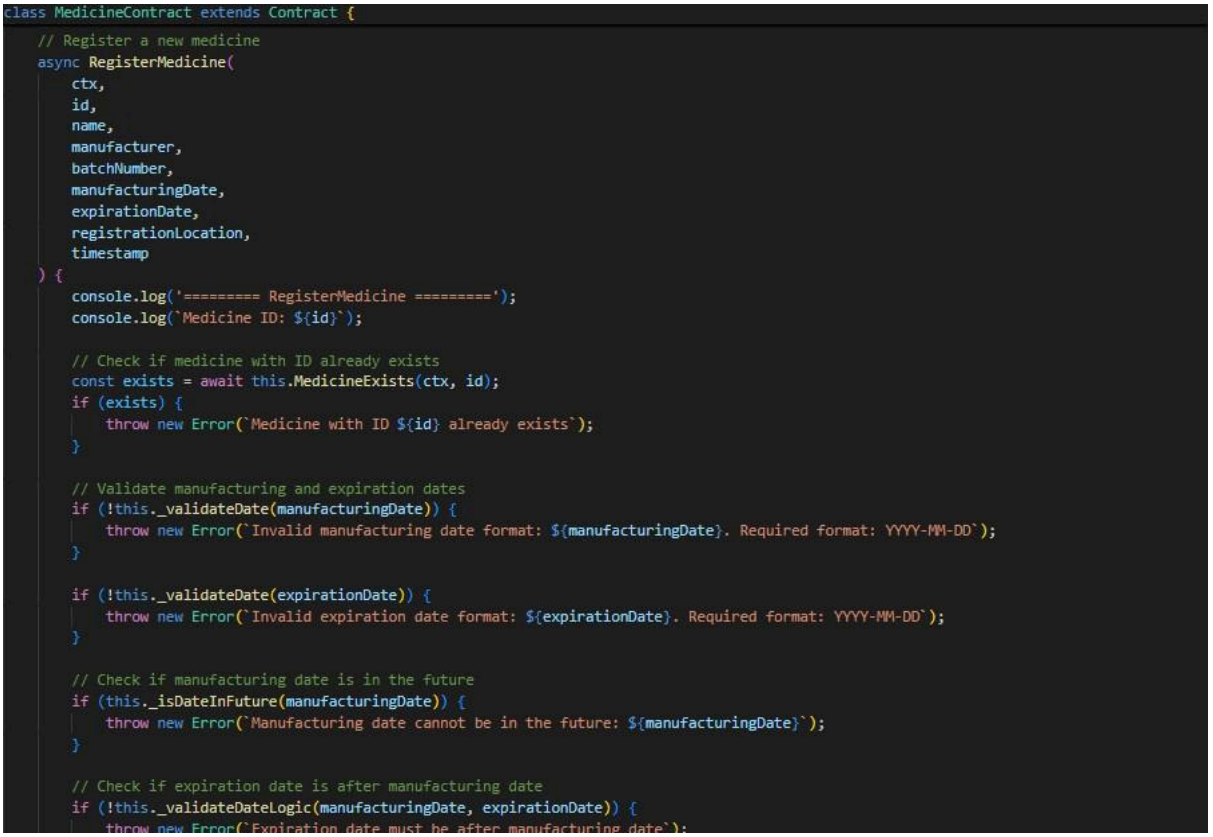
Code Snapshot: Express Route (Flag Medicine)
From fabric-test-app/routes/medicines.js

```
// @route   POST api/medicines/:id/flag
// @desc    Flag a medicine for issues
// @access  Private
router.post(
  '/:id/flag',
  [
    verifyToken,
    body('reason', 'Reason is required').not().isEmpty(),
    body('location', 'Location is required').not().isEmpty(),
  ],
  async (req, res) => {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
        console.log('Validation errors:', errors.array());
        return res.status(400).json({ errors: errors.array() });
      }

      const { id } = req.params;
      const { reason, location } = req.body;
      const flaggedBy = req.user.organization;
```

4.2.3. Blockchain Layer (Hyperledger Fabric)

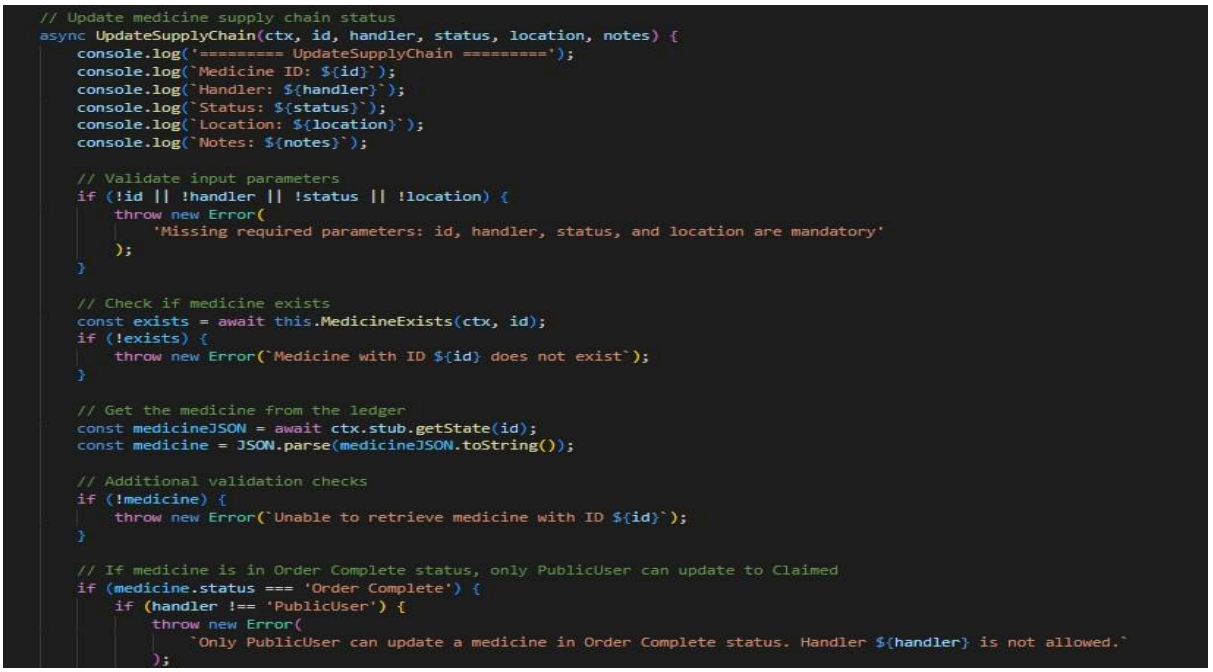
- Network: A Hyperledger Fabric network with multiple organizations (e.g manufacturers, distributors, regulators), each maintaining peers, orderers, and certificate authorities (CAs). Operates on a single channel (myChannel) for transaction privacy. The blockchain is initialised in Docker where containers act as nodes.
- Chaincode: Implemented in Node.js (medicine-contract.js), managing medicine lifecycle:
    - RegisterMedicine: Creates a new medicine record with QR code and supply chain entry.
    - UpdateSupplyChain: Updates medicine status (e.g., In Transit, Claimed) with role-based access control.
    - AssignDistributor: Assigns a distributor to a medicine.
    - FlagMedicine/UnflagMedicine: Flags medicines for issues (e.g tampering) and resolves them.
    - RecordScan: Logs QR code scans for auditability.
- Ledger: Stores immutable medicine records in CouchDB
- Consensus: Uses Raft for fault-tolerant transaction ordering.
- Security: MSP (Membership Service Provider) for identity management, TLS for secure communication.

Code Snapshot: Chaincode (Register Medicine)
From chaincode/medicine-contract/lib/medicine-contract.js



```
class MedicineContract extends Contract {
    // Register a new medicine
    async RegisterMedicine(
        ctx,
        id,
        name,
        manufacturer,
        batchNumber,
        manufacturingDate,
        expirationDate,
        registrationLocation,
        timestamp
    ) {
        console.log('========= RegisterMedicine =========');
        console.log(`Medicine ID: ${id}`);

        // Check if medicine with ID already exists
        const exists = await this.MedicineExists(ctx, id);
        if (exists) {
            throw new Error(`Medicine with ID ${id} already exists`);
        }

        // Validate manufacturing and expiration dates
        if (!this._validateDate(manufacturingDate)) {
            throw new Error(`Invalid manufacturing date format: ${manufacturingDate}. Required format: YYYY-MM-DD`);
        }

        if (!this._validateDate(expirationDate)) {
            throw new Error(`Invalid expiration date format: ${expirationDate}. Required format: YYYY-MM-DD`);
        }

        // Check if manufacturing date is in the future
        if (this._isDateInFuture(manufacturingDate)) {
            throw new Error(`Manufacturing date cannot be in the future: ${manufacturingDate}`);
        }

        // Check if expiration date is after manufacturing date
        if (!this._validateDateLogic(manufacturingDate, expirationDate)) {
            throw new Error(`Expiration date must be after manufacturing date`);
```

Code Snapshot: Chaincode (Update Supply Chain)
From chaincode/medicine-contract/lib/medicine-contract.js



```
    // Update medicine supply chain status
    async UpdateSupplyChain(ctx, id, handler, status, location, notes) {
        console.log('========= UpdateSupplyChain =========');
        console.log(`Medicine ID: ${id}`);
        console.log(`Handler: ${handler}`);
        console.log(`Status: ${status}`);
        console.log(`Location: ${location}`);
        console.log(`Notes: ${notes}`);

        // Validate input parameters
        if (!id || !handler || !status || !location) {
            throw new Error(
                'Missing required parameters: id, handler, status, and location are mandatory'
            );
        }

        // Check if medicine exists
        const exists = await this.MedicineExists(ctx, id);
        if (!exists) {
            throw new Error(`Medicine with ID ${id} does not exist`);
        }

        // Get the medicine from the ledger
        const medicineJSON = await ctx.stub.getState(id);
        const medicine = JSON.parse(medicineJSON.toString());

        // Additional validation checks
        if (!medicine) {
            throw new Error(`Unable to retrieve medicine with ID ${id}`);
        }

        // If medicine is in Order Complete status, only PublicUser can update to Claimed
        if (medicine.status === 'Order Complete') {
            if (handler !== 'PublicUser') {
                throw new Error(
                    `Only PublicUser can update a medicine in Order Complete status. Handler ${handler} is not allowed.`
                );
```

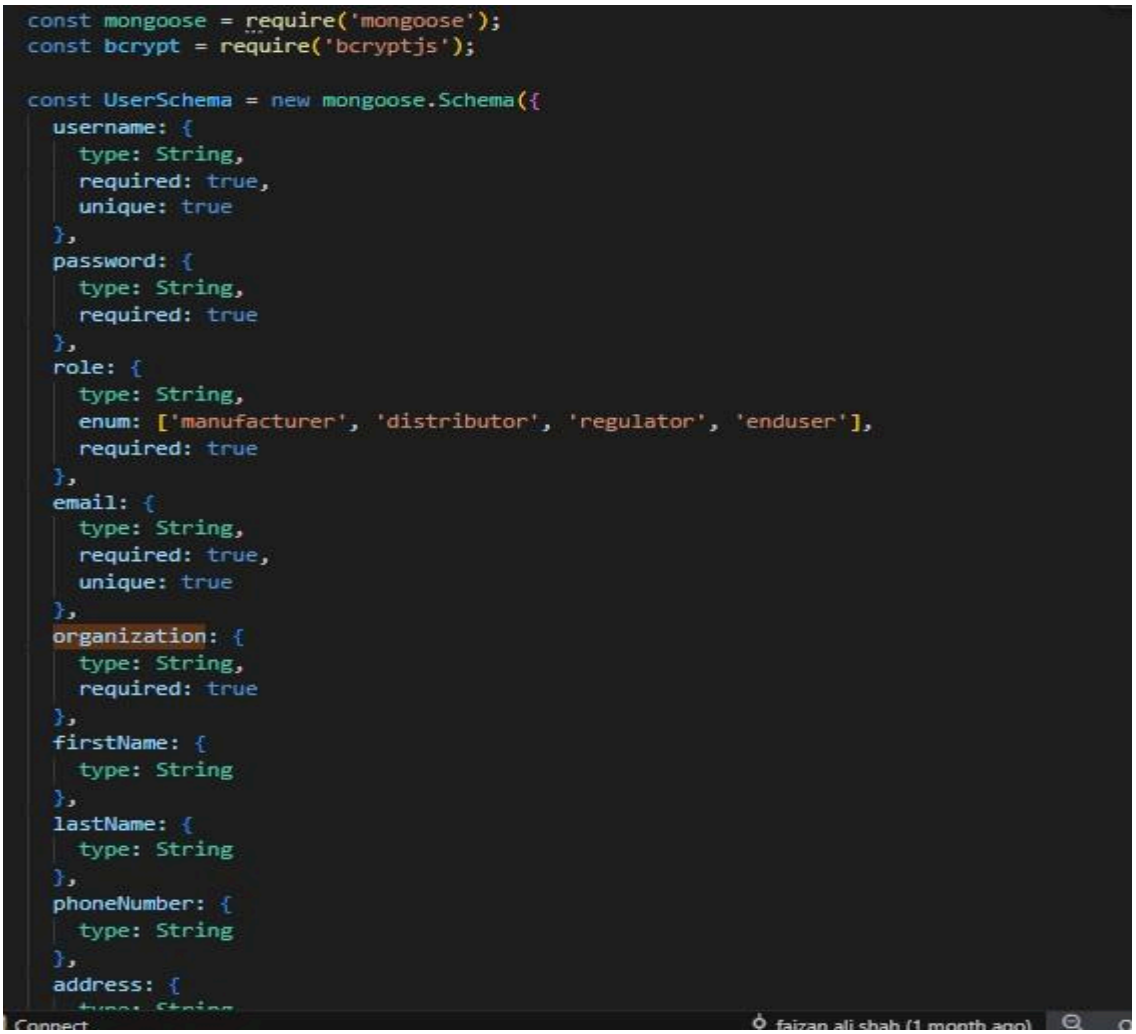
Code Snapshot: Chaincode (Flag Medicine)
From chaincode/medicine-contract/lib/medicine-contract.js

```
class MedicineContract extends Contract {

    // Flag medicine for issues (tampering, damage, etc.)
    async FlagMedicine(ctx, id, flaggedBy, reason, location) {
        if (!id || !flaggedBy || !reason || !location) {
            throw new Error(
                'Missing required parameters for flagging medicine'
            );
        }

        // Check if medicine exists
        const exists = await this.MedicineExists(ctx, id);
        if (!exists) {
            throw new Error(`Medicine with ID ${id} does not exist`);
        }

        // Get the medicine from the ledger
        const medicineJSON = await ctx.stub.getState(id);
        const medicine = JSON.parse(medicineJSON.toString());

        // Get transaction timestamp
        const txTimestamp = ctx.stub.getTxTimestamp();
        const milliseconds =
            txTimestamp.seconds.low * 1000 +
            Math.floor(txTimestamp.nanos / 1000000);
        const timestamp = new Date(milliseconds).toISOString();

        // Create a new supply chain entry for the flag
        const flagEntry = {
            timestamp,
            location,
            handler: flaggedBy,
            status: 'Flagged',
            notes: reason,
        };
```

4.4.4. Data Layer

- Off-Chain Storage: MongoDB stores user data, notifications, and manufacturer-distributor relationships to reduce blockchain load.
- Schemas:
    - User.js: Stores user credentials, roles, organization details (as a string), and registration hierarchy, with password hashing (bycrypt). Used for all user-related operations, including registration and authentication.
    - Notification.js: Stores in-system notifications for user communication.
- Integration: Mongoose for schema validation and querying, with indexes for performance (e.g unique email/username).

Code Snapshot: MongoDB Schema (User)
From fabric-test-app/models/User.js

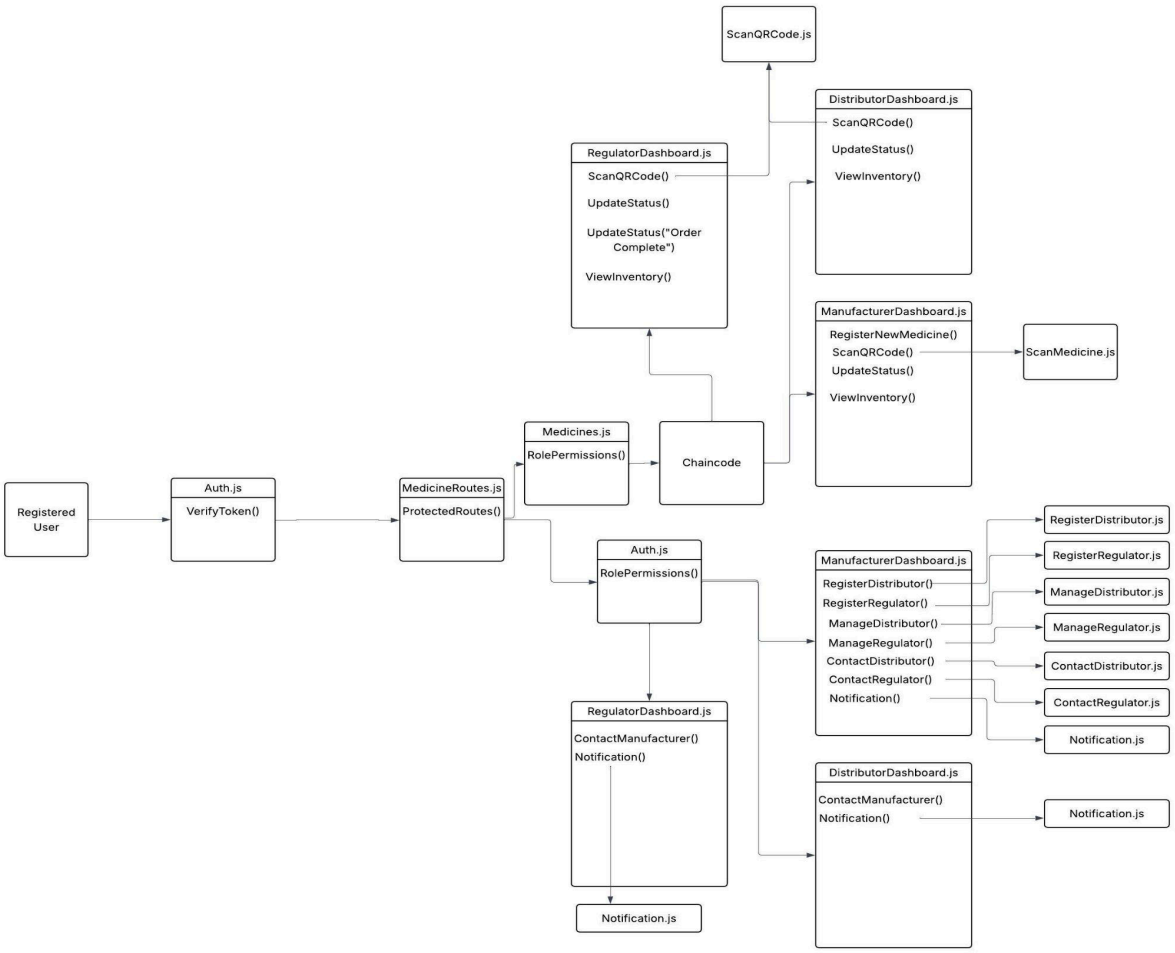


```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  role: {
    type: String,
    enum: ['manufacturer', 'distributor', 'regulator', 'enduser'],
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  organization: {
    type: String,
    required: true
  },
  firstName: {
    type: String
  },
  lastName: {
    type: String
  },
  phoneNumber: {
    type: String
  },
  address: {
```

Code Snapshot: MongoDB Schema (User)
From fabric-test-app/models/Notification.js

```
const mongoose = require('mongoose');

const notificationSchema = new mongoose.Schema({
  sender: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  recipient: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  senderOrganization: { type: String, required: true },
  recipientOrganization: { type: String, required: true },
  subject: { type: String, required: true },
  message: { type: String, required: true },
  relatedTo: { type: String, default: 'General' },
  medicineId: { type: String, default: null },
  isRead: { type: Boolean, default: false },
  isArchived: { type: Boolean, default: false },
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model('Notification', notificationSchema);
```

4.5 Communication System



5. Security Considerations

- Blockchain: MSP for identity management, TLS for peer communication, and chaincode-level RBAC.
- Backend: JWT authentication, input sanitization, and HMAC for QR code integrity.
- Data: MongoDB encryption at rest, TLS for data in transit, and bcrypt for password hashing.
- Frontend: CSRF protection and secure HTTP headers.
- QR Codes: Secure generation with HMAC signatures and expiration checks.

6. Role-Based Access Control

- Manufacturers: Self-register, register distributors/regulators, manage medicines (register, assign distributors, unflag).

- Distributors: Update supply chain status, verify medicines, communicate with manufacturers.
- Regulators: Monitor flagged medicines, approve compliance, access full supply chain data.
- End Users: Verify and claim medicines via public APIs without accounts.

7. Problems Faced and Solutions

Problem	Cause	Solution
Docker container not launching	Version conflict	Updated Docker Engine & fixed config
Chaincode not executing	Incorrect path	Refactored file structure
Peer certificates mismatch	Old wallet id being used	Create new wallet id's each time the blockchain is launched
Failing to create accurate wallet id's	Old connection-org1.json file	Update the connection-org1.json file each time the blockchain is launched

---

8. Testing and Results

- Testing Strategy: Stress testing, Unit testing, integration testing.
- Tools Used: Jest, Mohcta, Chai, Bash, Scripting.

9. Testing Methodology | Test Cases and Results

We did a variety of testing during development that were hard coded and after development where we utilised different libraries to understand the scope of the project as well as the things that we had missed.

A few things that we wanted to make sure we tested were:

- Stress testing the blockchain to figure out a breaking point to identify and improve the future expansions of the code and its current abilities to handle transactions and record them.
- Integration testing to make sure that the system architecture behaved the way we designed and the core functionality between the 3 layers, i.e React, NodeJS and Hyperledger fabric could communicate effectively with each other with no loopholes.

- Individual unit testing for each of the layers, establishing confidence in the robustness of the project and pointing out the things that were
    - Left out/Underdeveloped
    - Not delivering the full functionality that we wanted
    - Exploiting possible loopholes

9.1 Backend (Unit Testing)

Test Environment

- Framework: Jest
- Application: fabric-test-app
- Database: In-memory MongoDB
- Dependencies: Node.js, MongoDB Driver, SendGrid for notifications
- Test Directory: /fabric-test-app/__tests__
- Execution Command: npm test
- Execution Time: 34.39 seconds

Tests were organized into suites covering:

- Middleware: Authentication (auth.js) and role-based access control.
- Utilities: Password generation (passwordGenerator.js) and security functions (securityUtils.js).
- API Routes: Authentication (auth.js), medicine management (medicines.js), and notifications (notifications.js).

Tests were executed using **npm test** in the __tests__ directory. The application interacted with a Hyperledger Fabric network for blockchain operations , with mock tokens and users to simulate real-world scenarios.

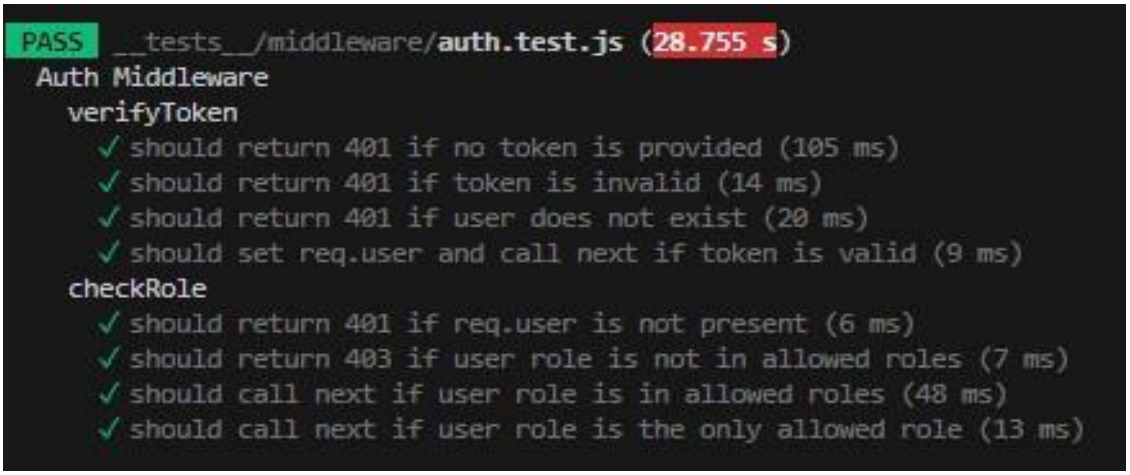
A total of 67 tests across 6 suites passed successfully in 34.39 seconds, covering middleware, utilities, and API routes. Below is a summary of each suite, with console output snapshots as code blocks and placeholders for image snapshots.

9.1.1. Auth Middleware (auth.test.js)

Tested the verifyToken and checkRole middleware for authentication and role-based authorization, ensuring secure access to API routes.

Key Tests:

- Returned 401 for missing or invalid tokens.
- Returned 401 if the user does not exist in the database.
- Set req.user and called next for valid tokens.
- Enforced role-based access (401/403 for unauthorized roles).

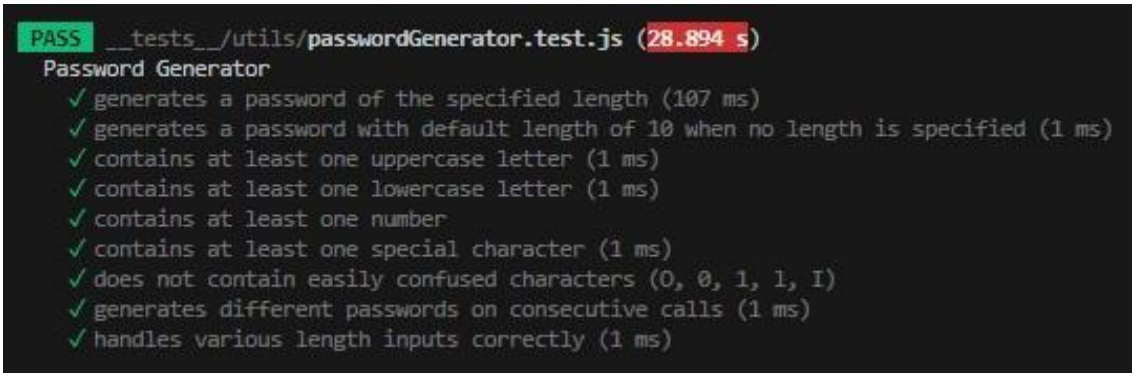


9.1.2. Password Generator (passwordGenerator.test.js)

Tested the password generation utility, ensuring secure and varied passwords for user accounts.

Key Tests:

- Generated passwords of specified lengths.
- Included uppercase, lowercase, numbers, and special characters.
- Avoid confusing characters.
- Produced unique passwords on consecutive calls

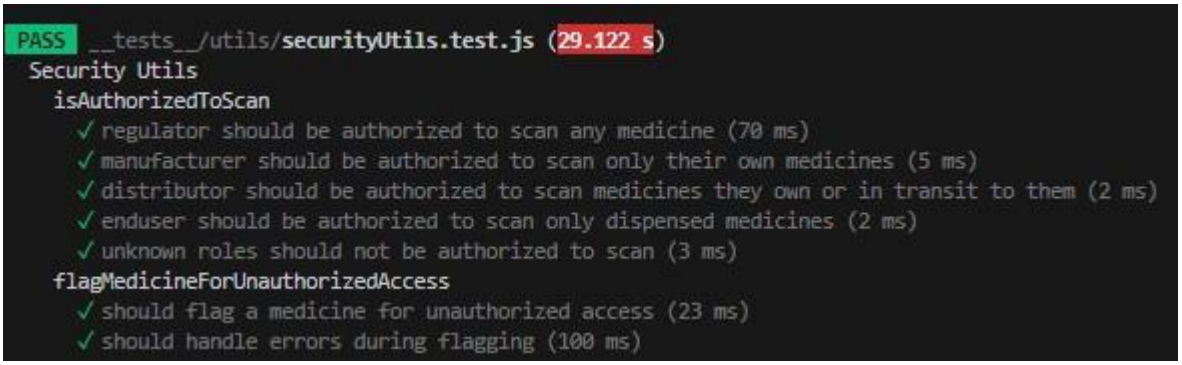


9.1.3. Security Utils (securityUtils.test.js)

Tested security functions for scan authorization, flagging unauthorized access, and recording security incidents, supporting the FlagMedicine.js component.

Key Tests:

- Authorized scans based on roles (regulator, manufacturer, distributor, enduser).
- Flagged medicines for unauthorized access, handling errors (e.g., wallet.get is not a function).



9.1.4. Notification Routes (notifications.test.js)

Tested API endpoints for creating and replying to notifications, supporting stakeholder communication.

Key Tests:

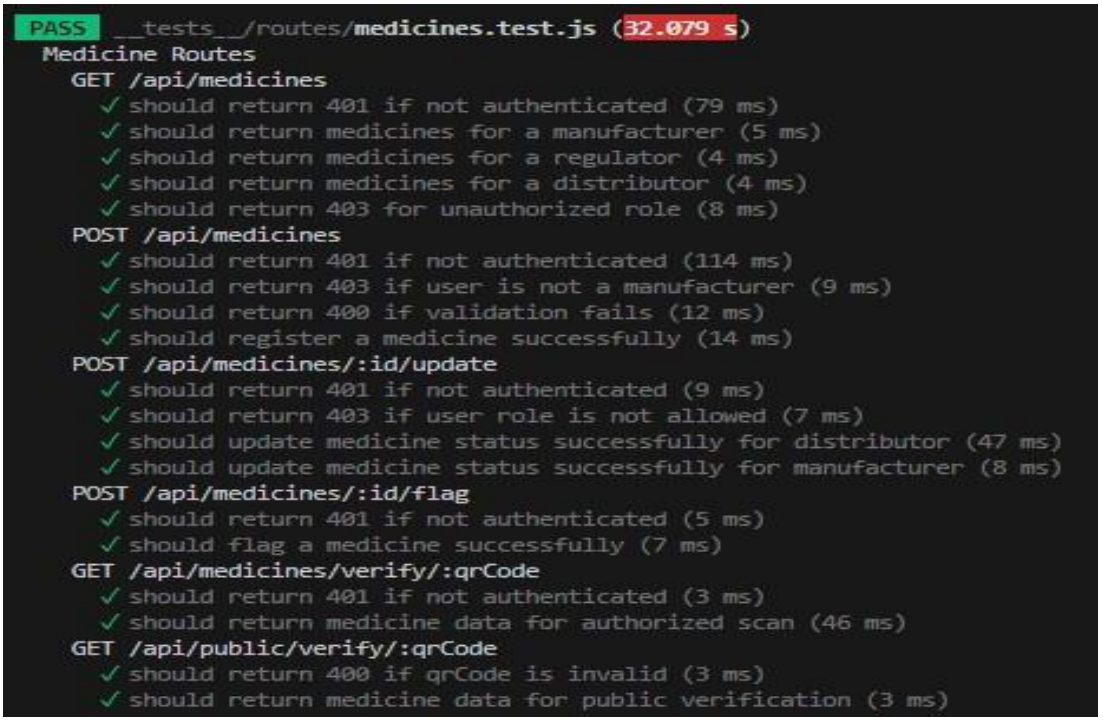
- Returned 401/403 for unauthenticated/unauthorized requests.
- Returned 400 for validation failures and 404 for missing users.
- Successfully created and replied to notifications (manufacturer ↔ distributor).
- Encountered errors (Notification is not a constructor) during notification creation.

9.1.5. Medicine Routes (medicines.test.js)

Tested API endpoints for managing medicines, including retrieval, registration, updates, flagging, and QR code verification, directly supporting FlagMedicine.js and MedicineStatus.js.

Key Tests:

- Returned 401/403 for unauthenticated/unauthorized requests.
- Successfully retrieved, registered, updated, and flagged medicines.
- Verified medicines via QR codes for authorized and public users.
- Logged parameters for updates and flagging.

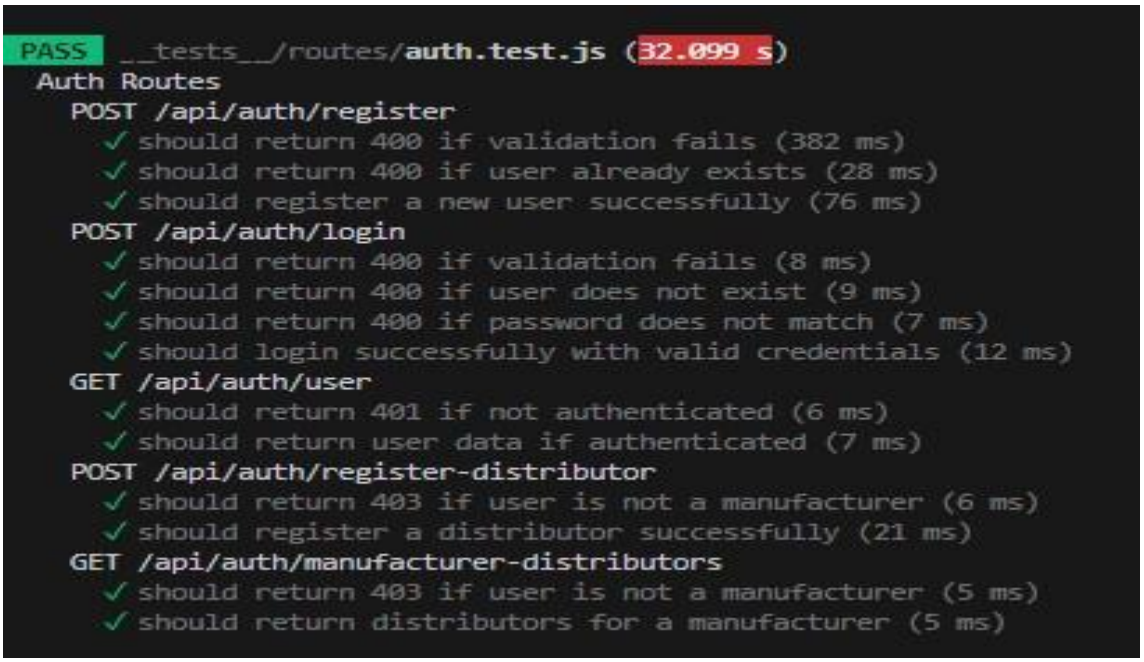


9.1.6. Auth Routes (auth.test.js)

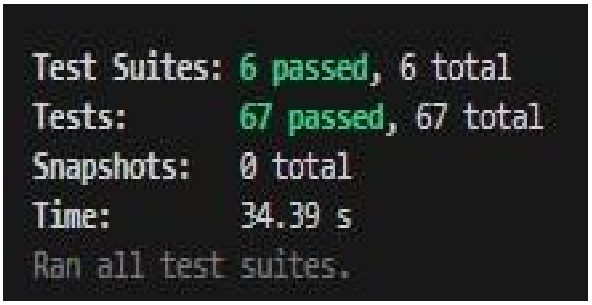
Tested authentication endpoints for user registration, login, and distributor management.

Key Tests:

- Returned 400 for validation failures or existing users.
- Successfully registered and logged in users.
- Returned user data for authenticated requests.
- Registered distributors for manufacturers.



All test cases were passed for Backend Unit Testing



9.2 Hyperledger Fabric (Unit Testing)

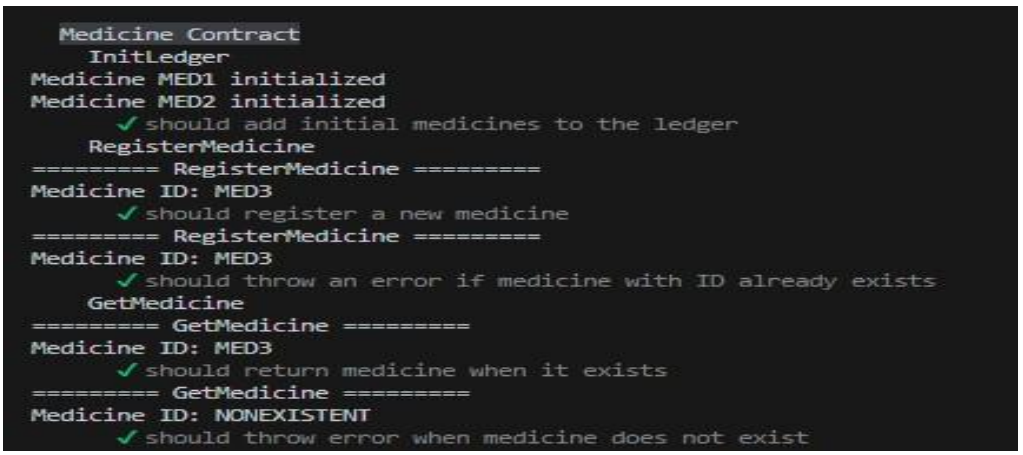
Test Environment

- Framework: Mocha (with Chai for assertions)
- Chaincode: medicine-contract (version 1.0.0)
- Network: Hyperledger Fabric (development mode)
- MSP: Org1MSP (with mock MSP for edge case testing)
- Test Directory: /fabric-samples/chaincode/medicine-contract/test
- Timeout: 5000ms per test
- Execution Command: npm test

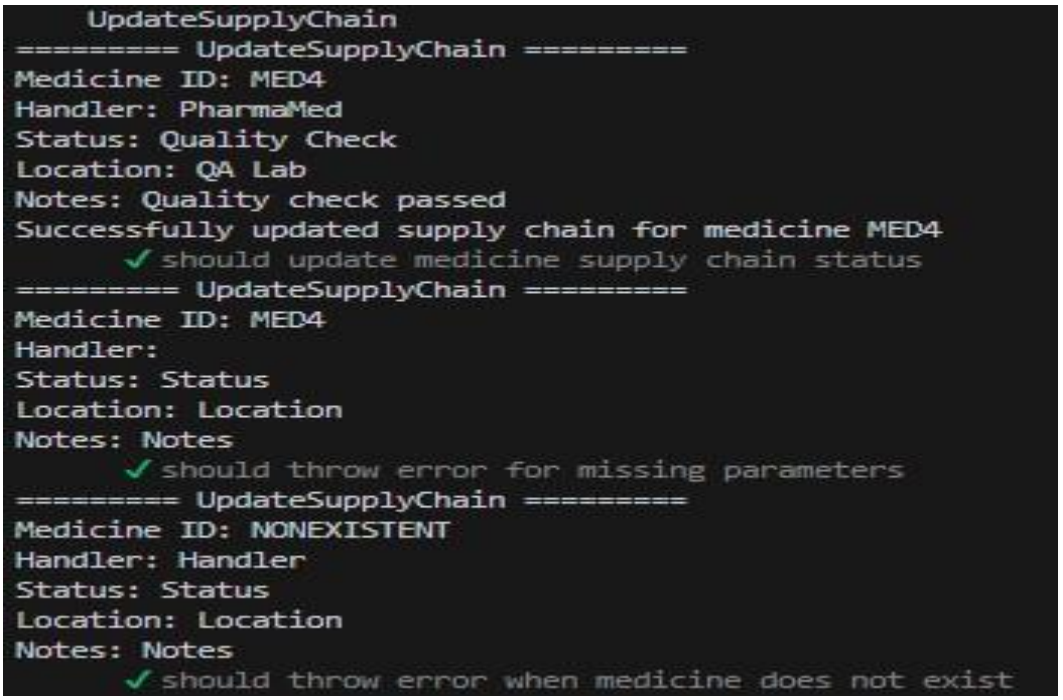
A total of 35 unit tests were executed, all of which passed successfully with an execution time of 156ms. The tests covered the following key areas:

9.2.1 Core Functions

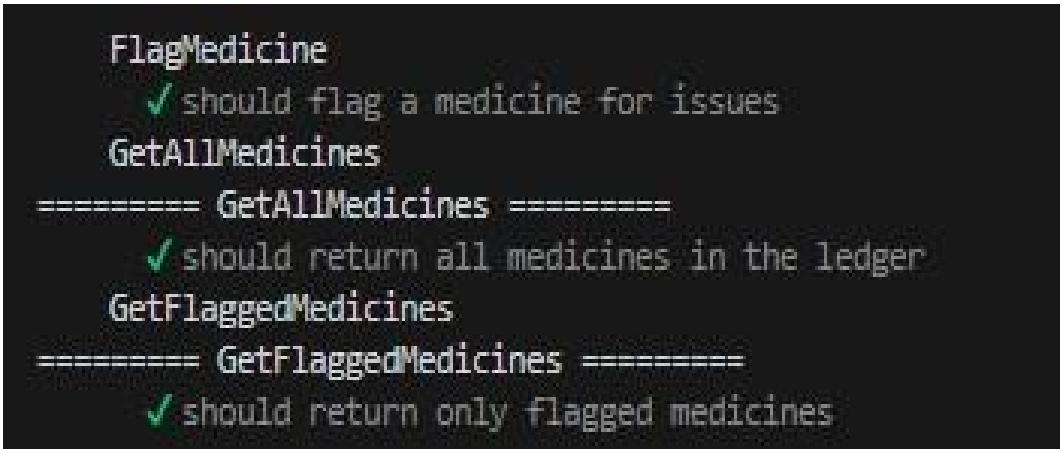
- InitLedger: Successfully initialized the ledger
- RegisterMedicine: Successfully registered a new medicine and correctly threw an error when attempting to register a duplicate ID.
- GetMedicine: Successfully retrieved existing medicine details and threw an error for non-existent medicines (NONEXISTENT).



- UpdateSupplyChain: Successfully updated the supply chain status for a medicine and threw errors for missing parameters or non-existent medicines.



- FlagMedicine: Successfully flagged a medicine for issues, ensuring it was marked as problematic in the ledger.
- GetAllMedicines: Successfully retrieved all medicines in the ledger.
- GetFlaggedMedicines: Successfully returned only flagged medicines.



9.2.2 Advanced Functions

- UpdateEnhancedSupplyChain: Successfully updated a medicine with enhanced supply chain data
- RecordScan: Successfully recorded a scanning activity for a medicine

```
    AssignDistributorsToMedicine
Current MSP ID: Org1MSP
Auth Check: { clientMSP: 'Org1MSP', clientOrg: 'Org1', manufacturer: 'PharmaMed' }
DEVELOPMENT MODE: Bypassing manufacturer check for PharmaMed
      ✓ should assign distributors to a medicine
      ✓ should throw error when medicine does not exist
Current MSP ID: Org1MSP
Auth Check: { clientMSP: 'Org1MSP', clientOrg: 'Org1', manufacturer: 'PharmaMed' }
DEVELOPMENT MODE: Bypassing manufacturer check for PharmaMed
      ✓ should throw error with invalid distributors JSON
    UpdateEnhancedSupplyChain
========= UpdateEnhancedSupplyChain =========
Medicine ID: MED1
New Status: Quality Checked
Phase: QualityAssurance
      ✓ should update medicine with enhanced supply chain data
    RecordScan
========= RecordScan =========
Medicine ID: MED1
Scanned by: HealthCare Clinic (Pharmacist: Dr. Smith)
Location: Dublin Clinic
Successfully recorded scan for medicine MED1
      ✓ should record scanning activity in medicine history
```

9.2.3 Supply Chain Scenarios

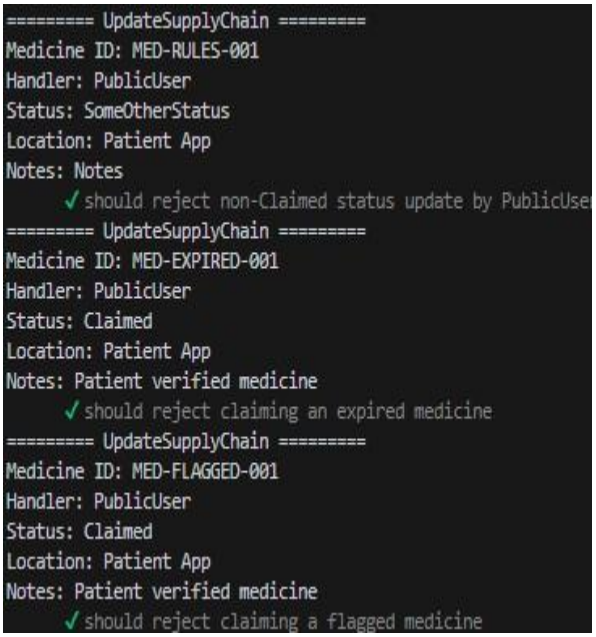
- End-to-End Supply Chain Flow: Successfully tracked a medicine through its entire supply chain journey, From:
    - Registration
    - Status Update
    - Distribution
    - Pharmacy/end user delivery

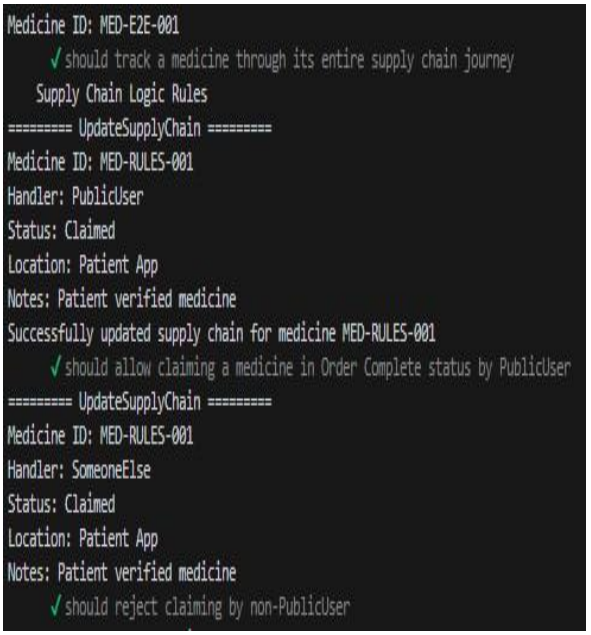
```
========= UpdateSupplyChain =========
Medicine ID: MED-E2E-001
Handler: National Distributor
Status: In Distribution
Location: Distribution Center, Cork
Notes: Received at central distribution
Successfully updated supply chain for medicine MED-E2E-001
========= GetMedicine =========
Medicine ID: MED-E2E-001
========= GetMedicine =========
Medicine ID: MED-E2E-001
========= UpdateEnhancedSupplyChain =========
Medicine ID: MED-E2E-001
New Status: Repackaged
Phase: Remediation
========= GetMedicine =========
Medicine ID: MED-E2E-001
========= UpdateSupplyChain =========
Medicine ID: MED-E2E-001
Handler: City Pharmacy
Status: At Pharmacy
Location: Main Street Pharmacy, Dublin
Notes: Delivered to pharmacy
Successfully updated supply chain for medicine MED-E2E-001
```

```
  Medicine Contract - Supply Chain Scenarios
    End-to-End Supply Chain Flow
========= RegisterMedicine =========
Medicine ID: MED-E2E-001
========= GetMedicine =========
Medicine ID: MED-E2E-001
========= UpdateSupplyChain =========
Medicine ID: MED-E2E-001
Handler: PharmaCo Ltd
Status: Quality Check
Location: QA Lab, Dublin
Notes: All quality checks passed
Successfully updated supply chain for medicine MED-E2E-001
========= GetMedicine =========
Medicine ID: MED-E2E-001
Current MSP ID: MockMSP
Auth Check: {
  clientMSP: 'MockMSP',
  clientOrg: 'Mock',
  manufacturer: 'PharmaCo Ltd'
}
```

    - Dispensing (Order Complete).
    - Patient verification (Claimed via Patient App).

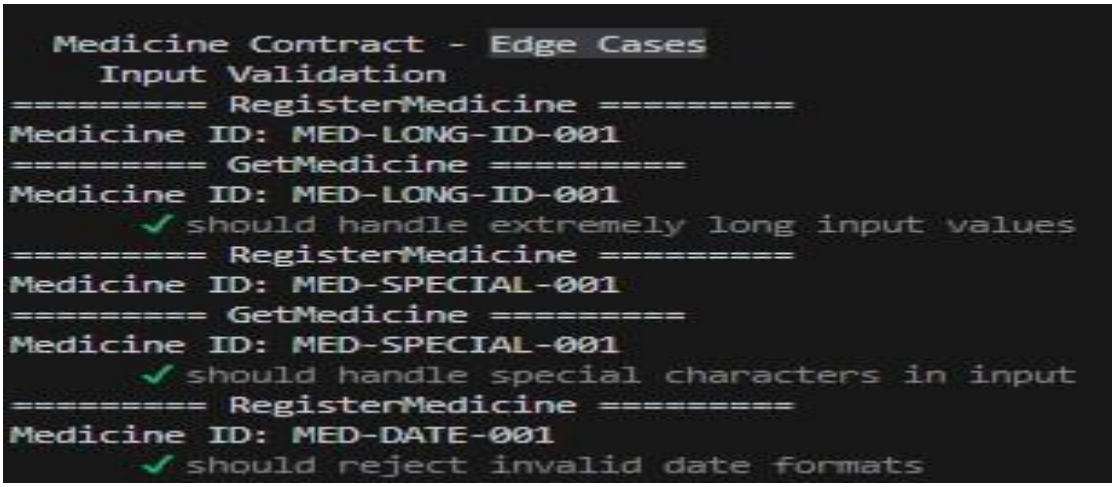
- **Supply Chain Logic Rules**:
    - Allowed public users to claim medicines in Order Complete status
    - Rejected claims by non-public users.
    - Rejected non-Claimed status updates by public users.
    - Rejected claims for expired or flagged medicines.
    - Rejected updates to already claimed medicines.



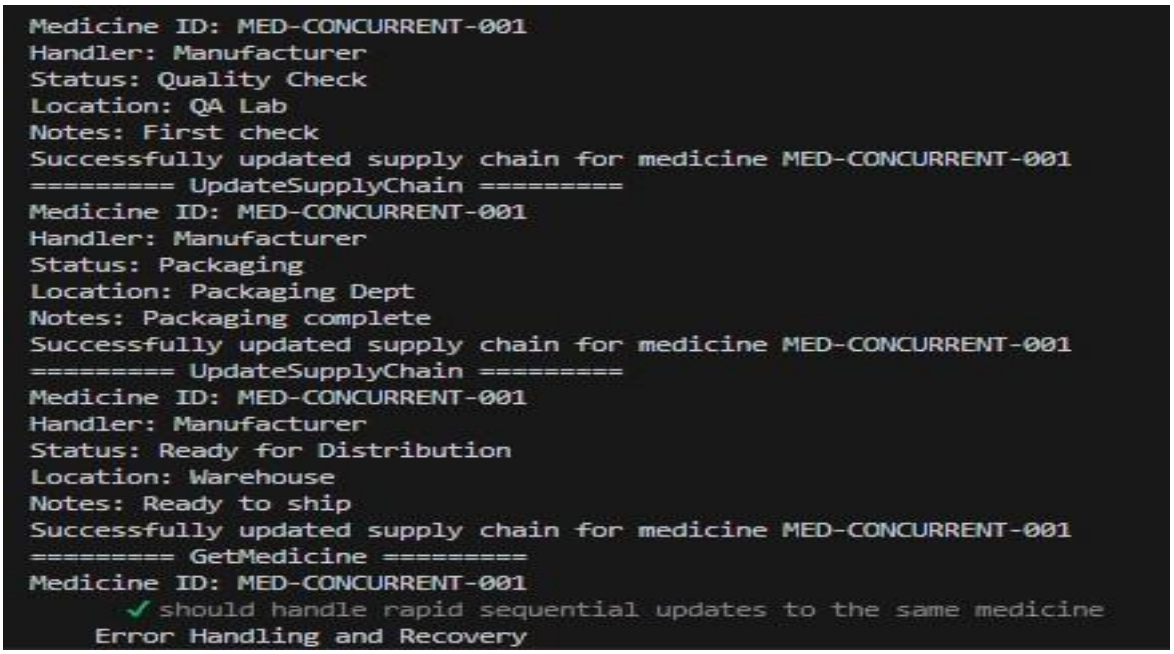


9.2.4. Edge Cases

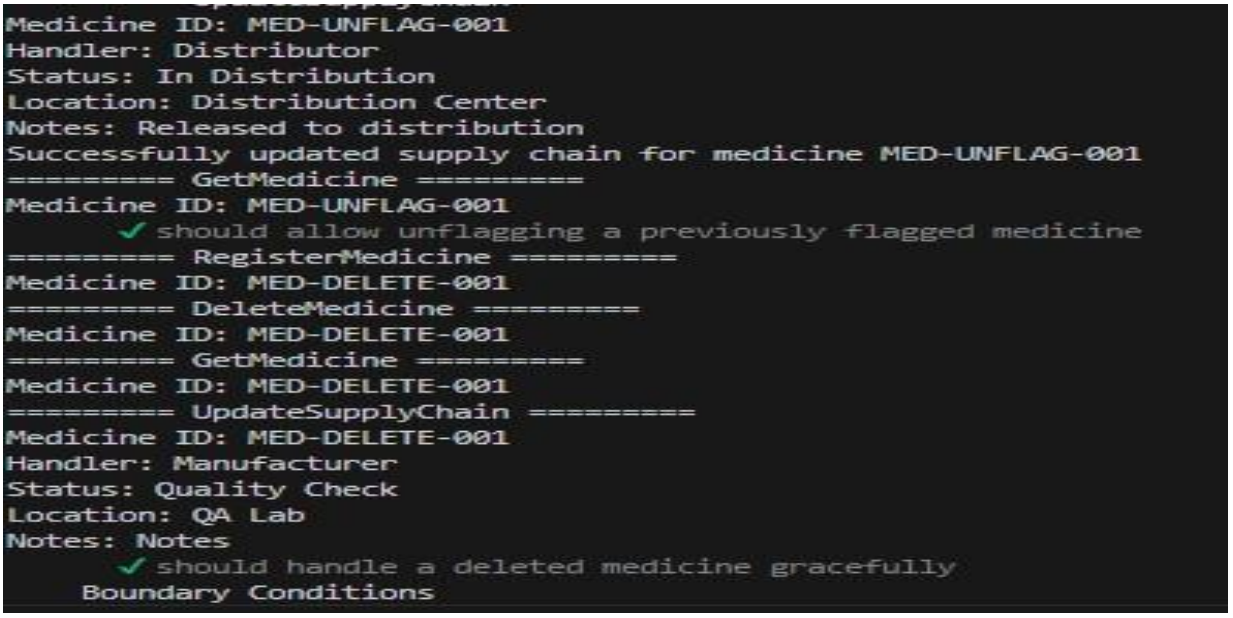
- **Input Validation**:
    - Handled extremely long input values.
    - Handled special characters in input.
    - Rejected invalid date formats.



- Concurrent Modifications: Successfully handled rapid sequential updates to the same medicine (MED-CONCURRENT-001, statuses: Quality Check, Packaging, Ready for Distribution).

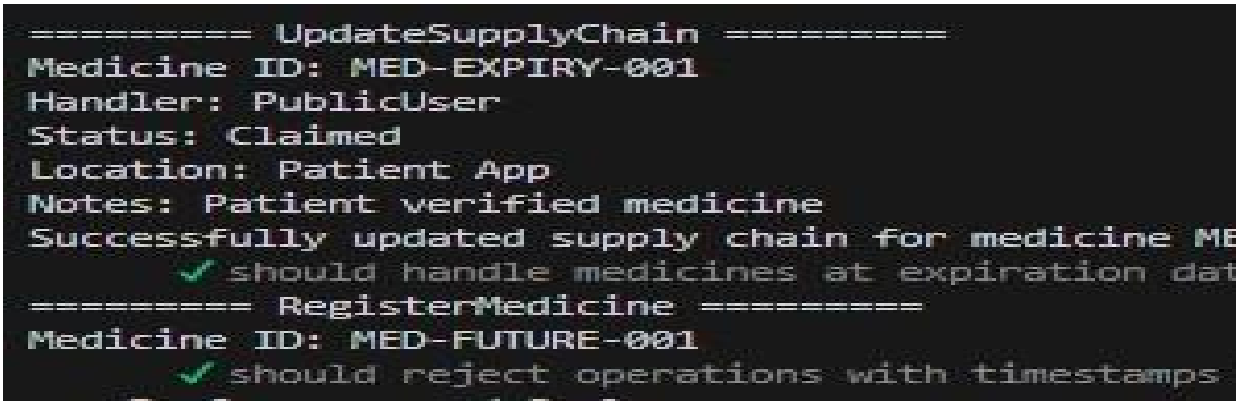


- **Error Handling and Recovery**:
  - Successfully unflagged a previously flagged medicine and allowed subsequent updates.
  - Gracefully handled operations on a deleted medicine.

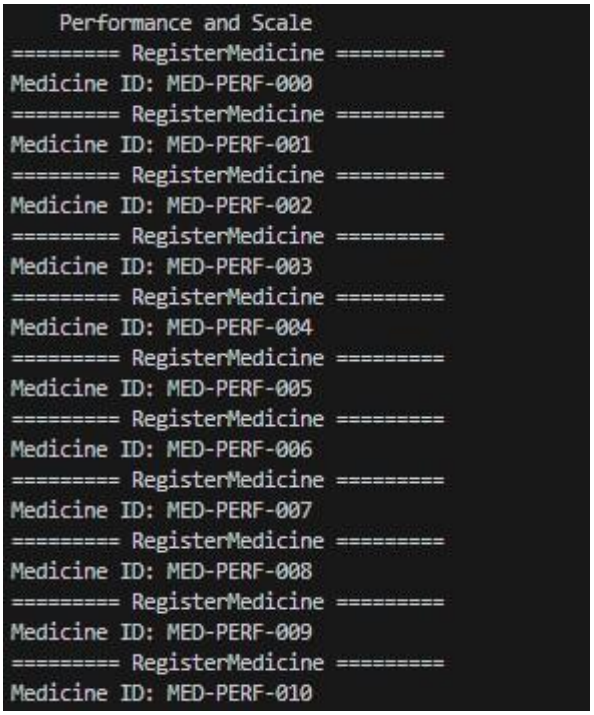


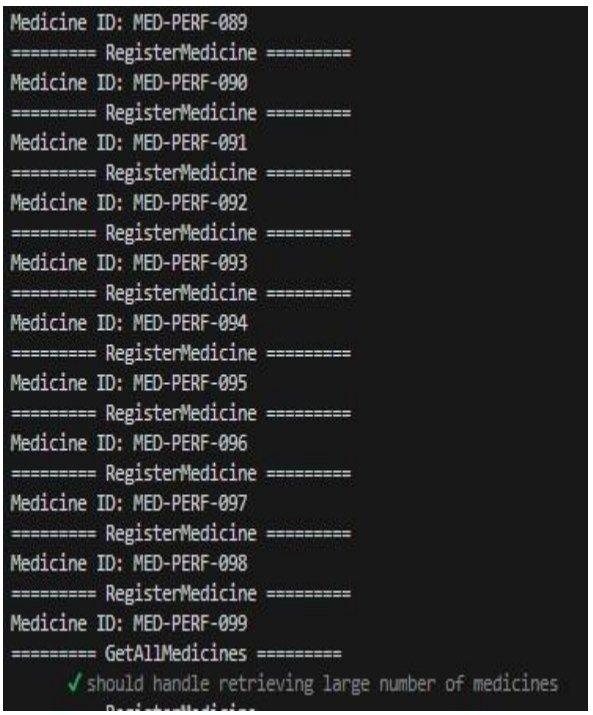
- **Boundary Conditions**:
  - Successfully processed medicines at expiration date boundaries (MED-EXPIRY-001, statuses: Ready for Distribution, At Pharmacy, Order Complete, Claimed).

- Rejected operations with future timestamps (MED-FUTURE-001).



- **Performance and Scale**:
  - Successfully registered and retrieved 100 medicines 0-100.
  - Handled a medicine with a large supply chain history (MED-HISTORY-001, 50 scan records, completed in 39ms).





Test Coverage

The tests achieved comprehensive coverage of the chaincode's smart contract functions, including:

- All core CRUD operations (create, read, update, delete).
- Advanced supply chain management features
- Edge cases and error conditions (e.g invalid inputs, concurrent updates, deleted medicines).

- Performance under scale
- Authorization and supply chain logic rules.

No test failures were reported

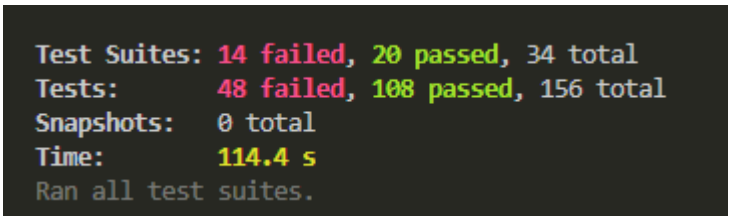
Conclusion

The unit testing phase successfully validated the medicine-contract chaincode, with all 35 tests passing in 156ms. The tests confirmed that the chaincode correctly handles core and advanced supply chain operations, enforces logic rules, and manages edge cases effectively. The results provide confidence in the chaincode's reliability for managing medicine supply chain data, supporting client-side features like flagging and status tracking.

9.3. Unit Testing (Frontend)

Test Environment

- Framework: Jest
- Application: react-frontend (frontend)
- Server: http://localhost:3001
- Test Directory: /reac-frontend/src/components/dashboard/__tests__/integration
- Execution Command: npm test



- Integral parts and pages from the frontend passed the tests which include the core blockchain functionality that was extended to the backend and then the frontend.
- Backend supporting functions that expanded on the core functionalities like delivery history, flag medicines etc. also passed all the tests.
- The test cases failed at two crucial points, that is the AssignDistributor functionality and NotificationForm.js.

Break Down of The Failed Test Cases:

- AssignDistributor: This functionality describes how the manufacturer can assign specific medicines to specific distributors. This allowed the distributor to only view medicine in its inventory that was assigned to them.

    - Now the only part working from the functionality is that the distributor can only view medicine in it's inventory, once the regulator receives the medicine, the distributor can no longer view those medicines.
    - Assigning the actual distributor to the medicine is not working as we did not prepare for future updates like this one. The assign distributor was supposed to be implemented on the manufacturer dashboard pages as well as distributor inventory pages that allowed for additional functionality but due to time constraints these updates seems hard to implement.
- NotificationForm.js: Although this file has complete functionality in terms of what it is designed for, a few design flaws that were ignored during the early development of the website like temporary hard coded unit testing and error handling.

9.4 Integration testing (Blockchain - Backend)

Test Environment

- Tool: Bash script (integration-tests.sh) with curl
- Application: fabric-test-app (backend)
- Server: http://localhost:3000
- Database: In-memory MongoDB
- Blockchain: Hyperledger Fabric (interfaced via chaincode)
- Test Directory: /fabric-test-app/__tests__/integration
- Execution Command: ./integration-tests.sh

The integration test suite executed 16 test steps, all completing successfully, as indicated by the script's output. Below is a summary of the test cases, with console output snapshots for key steps and placeholders for image snapshots.

The test was conducted using a Bash script (integration-tests.sh) that automates API calls with curl, a command-line tool for making HTTP requests. The script:

- Simulates a User Workflow: It mimics a user (e.g a manufacturer) performing actions like logging in, registering medicines, updating supply chain status, flagging issues, verifying medicines, and managing notifications.
- Uses Authentication: A test user (testuser1@example.com, password: password123) logs in to obtain a JSON Web Token (JWT), which is included in subsequent requests for authenticated endpoints.
- Exits on Failure: If any step fails, the script terminates with an error message.

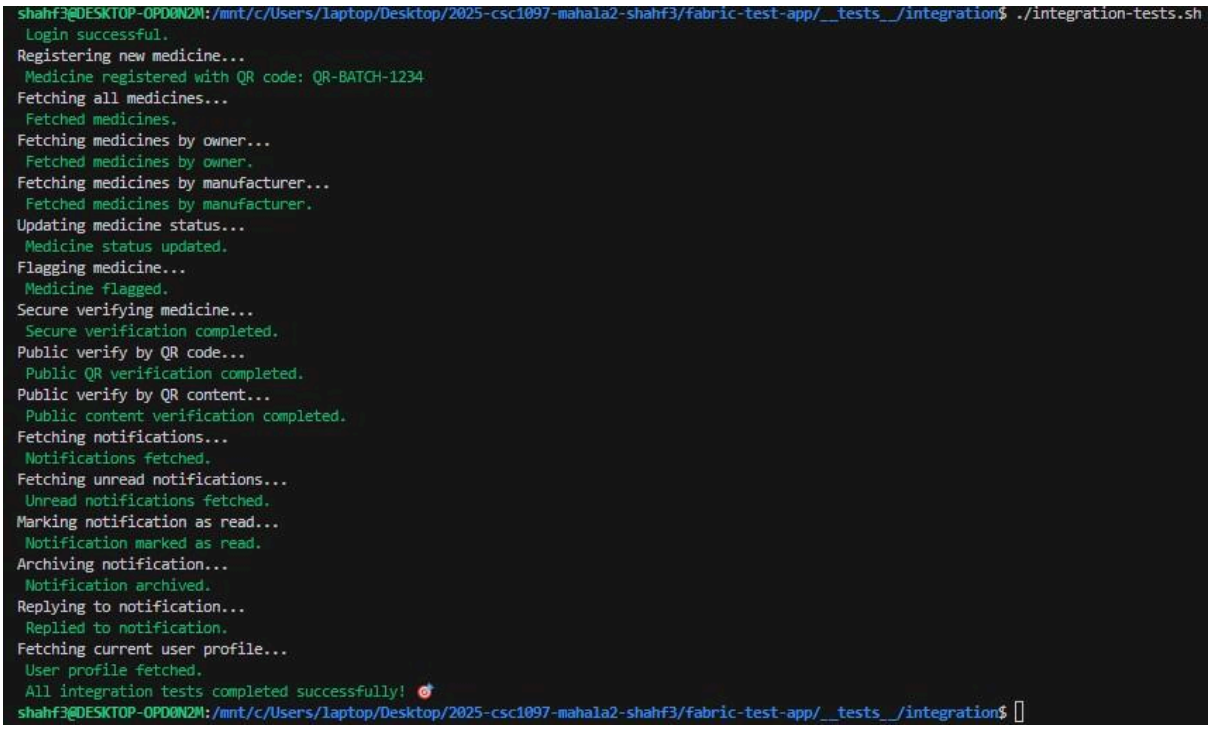
Execution Process

The script was executed in the /fabric-test-app/__tests__/integration directory using the command:

./integration-tests.sh

Test Summary

Description	Outcome
1. Login	Successful login, token acquired for subsequent requests.
2. Register a New Medicine Regisfered a medicine (MED-010, Paracetamal, BATCH-1234) via / api /medicines, generating a QR code (QR-BATCH-12340).	Medicine registered, QR code generated for verification
3. Fetch All Medicines	Medicines fetched successfully
4. Fetch Medicines by Owner	Owner-specific medicines fetched
5. Fetch Medicines by Manufacturer via /api/medicines/owner/restOrgain i-zation	Manufacturer-specific medicines fetched
6. Update Medicine Supply Chain Updated the supply chain status smf MED-001 to In Transit at Warehouse. Du-blin via /api/medicines/MED-001/update	Status updated successfully
7. Flagging the Medicine	Medicine flagged successfully
8. Secure verify by QR (Authenficated) Verified a medicine via QR code (QR-B-ATCH-1234) using /api/medicines/verify $QR_CODE with authentication	Secure verification completed
9. Public Verify by QR Code	Public QR verification completed
10. Public Verify by QR Content	Public content verification completed
11. Get Notifications	Notifications fetched successfully
12. Fetch Unread Notifications Count	Notifications fetched fetched
13. Mark Notification as Read	Notification marked as read
14. Archive Notification	Notification archived
15. Reply to Notification	Reply sent successfully
16. Fetch Logged-in User Profile	Profile fetched successfully



The tests covered critical end-to-end workflows:

- User authentication and profile management.
- Medicine registration, supply chain updates, flagging, and verification.
- Notification creation, retrieval, and management.
- Integration with Hyperledger Fabric chaincode and MongoDB persistence.

Conclusion

The integration tests successfully validated 17 critical workflows of the fabric-test-app, confirming robust functionality for user authentication, medicine management, flagging, verification, and notifications. The tests align with the FlagMedicine.js and MedicineStatus.js components and integrate with the Hyperledger Fabric chaincode.

9.5 Blockchain Stress Testing

The stress testing process was fully automated through Javascript scripts. The following scripts were used:

- enhanced-tests.js:
    - Purpose: Executes stress tests against the medicine-contract chaincode on the Hyperledger Fabric network.
    - Functionality: Defines test profiles (Mixed Workload Test, Endurance Test) with configurable parameters (e.g., transaction mix, concurrency, preload data). It uses the fabric-network SDK to interact with the network, submitting

and evaluating transactions (RegisterMedicine, UpdateSupplyChain, GetMedicine, GetAllMedicines,FlagMedicin).
    - The script records performance metrics (TPS, latency, success/failure counts)
- test-automation.js:
    - Purpose: Automates the execution of multiple test profiles sequentially.
    - Functionality: Parses command-line arguments to select profiles (e.g., node test-automation.js mixed endurance), spawns child processes to run enhanced-tests.js for each profile, and manages timeouts (default: 10 minutes). It ensures output directories exist, logs test progress, and optionally triggers report generation.
- visualization-tools.js:
    - Purpose: Processes test results and generates HTML reports for analysis.
    - Functionality: Scans the ./test-results directory for JSON result files, processes metrics (TPS, latency, success rate), and generates detailed HTML reports for each test run. It also creates a summary report (test-results-summary.html) with a performance comparison chart.

Test Name	Total Tx	Success Tx	Failed Tx	Tx Success Rate	TPS	Min Latency (ms	Avg Latency (ms	Median Latency	P95 Latency (ms)	Max Latency (ms)
Mixed Workload Test	60,000	2,359	57,641	3.93%	5.93	27,384	52,359.82	50,233	73,156	79,570
Mixed Workload Test	60,000	2,359	57,641	3.93%	5.93	27,384	52,359.82	50,233	73,156	79,570
Mixed Workload Test	30,000	8,980	21,020	29.93%	34.18	27,020	57,504.05	55,258	79,357	87,541
Mixed Workload Test	25,000	4,613	20,387	18.45%	21.38	9,462	31,136.46	32,848	47,367	59,625
Mixed Workload Test	20,000	5,461	14,539	27.30%	26.76	8,986	37,922.53	33,412	82,878	88,345
Mixed Workload Test	10,000	4,039	5,961	40.39%	36.11	12,687	27,074.05	27,402	45,566	74,045
Mixed Workload Test	8,000	4,528	3,472	56.60%	44.28	9,410	31,017.22	29,608.50	61,292	71,601
Mixed Workload Test	4,000	3,599	401	89.98%	42.71	208	1,870.38	1,616	3,604	5,086
Endurance Test	2,037	2,023	14	99.31%	33.63	15	78.97	71	163	277
Endurance Test	2,037	2,023	14	99.31%	33.63	15	78.97	71	163	277
Mixed Workload Test	2,000	1,822	178	91.10%	73.29	191	1,090.88	985.5	2,505	5,523
Mixed Workload Test	1,000	965	35	96.50%	85.28	183	760.79	633	1,505	2,843
Mixed Workload Test	1,000	999	1	99.90%	73.57	33	192.15	185	382	622
Mixed Workload Test	400	400	0	100.00%	79.22	45	169.37	159.5	332	376
Mixed Workload Test	200	200	0	100.00%	25.62	28	656.25	469.5	1,803	2,531
Mixed Workload Test	200	200	0	100.00%	16.59	52	985.88	681	3,036	3,547
Mixed Workload Test	150	150	0	100.00%	27.36	27	367.76	193	952	1,612
Mixed Workload Test	80	79	1	98.75%	18.22	17	193.41	84	761	1,026

Total Transactions

The total number of blockchain transactions submitted during the test run, whether they succeeded or failed.

Successful Transactions
The number of transactions that completed without error and were correctly processed by the network.

Failed Transactions
The number of transactions that failed

Transactions Per Second (TPS)
Calculated as `successful` **transactions ÷ total test duration** `(seconds)`.
TPS is a key performance indicator reflecting system throughput, how many operations per second the network can handle under test conditions.

Latency Metrics
These measure how long individual transactions take from submission to confirmation.

- Average Latency: Mean time across all successful transactions.
    - Helps assess general system responsiveness.
- Minimum Latency: Fastest observed transaction time.
    - Shows the best-case performance.
- Median Latency: Middle value when all latencies are sorted.
- Maximum Latency: Longest observed transaction time.
    - Important for understanding worst-case delays under load.
- 95th Percentile Latency (P95): 95% of transactions completed within this time.

Success Rate
Percentage of successful transactions out of the total

((success ÷ (success + failure)) × 100).

Provides a simple view of system reliability under stress.

Conclusion

The detailed stress test results for the 19 runs reveal that the medicine-contract chaincode performs reliably up to 4,000 transactions, achieving up to 85.28 TPS, 169.37 ms average latency, and 100% success rate. However, at higher volumes (8,000–60,000 transactions), performance degrades significantly, with success rates dropping to 3.93%, TPS falling to 5.93, and latency spiking to 57,504.05 ms.

10. Future Work

We are very excited to continue our work on this project and proud of what we have built. We have discussed the direction we want to take in the future which includes the features we talked about that we could not implement due time constraints. Additionally we would like to:

- Test and deploy our blockchain to actual individual peers instead of docker containers. Currently every time the project is launched, it starts fresh with no prior history of the assets. Deploying it full time we can further develop and test the full potential of Hyperledger Fabric.
- Deploy our website and make it connect and communicate with deployed blockchain to test, develop and simulate a more real-life environment so we can learn more about the possibilities and constraints of not only the project but also the technology we used.
- Learn more about supply chain software infrastructure and stakeholder needs to use as a guide for us that will help get a clearer vision of a consumer ready product that can be used in the real world.

11. Conclusion

Finishing the project we achieved a working ecosystem with roles that compliment, monitor and update statuses that help streamline the whole supply chain process with accountability on every action taken. These actions can ultimately be monitored by the manufacturer role ensuring the supply chain route and process details are only known to them, ensuring privacy for the manufacturers.

We are proud to say that our system mitigates any chances of an asset being replicated and provides full confidence to the end-user in being the first consumer to own the medicine. The use of profile creation via the manufacturer that sends the account details to the enrolled user(Distributors and Regulators) their account log in's and password via email using the SendGrid API. We also implemented an in-built messaging system that sends notifications to the respected users.

The blockchain aspect of the project had a very steep learning curve which made us underestimate ourselves in terms of the functionalities that we could deliver which hindered how we implemented the functionalities over what we had promised in the functional specification.

Some of these additional features were managing 2 separate QR codes for a medicine and a batch of medicines that would be used for the supply chain monitoring and tracking the asset. We could only implement the QR code for the medicine and use it for both tracking the asset through the supply chain and also to be claimed by the end user. This was due to unclear planning and lack of time.

Additionally, we wanted to allow the manufacturer role to be able to define the supply chain route for each medicine and assign distributors respectively. Implementing this functionality from the beginning turned out to be very time consuming and hard to implement on top of the existing code. More specifically, assigning distributor to a medicine was only resulting in the same distributor being assigned to the medicine no matter what the manufacturer role chose. This occurred due to the design of the system we implemented where we did not incorporate an option for future changes like these.

Although we are proud that we delivered what we promised on the functional specification and some more, we believe we could have taken this project to the next level if we could have implemented these additional functionalities.

12. References

https://ieeexplore.ieee.org/abstract/document/9431789

https://pmc.ncbi.nlm.nih.gov/articles/PMC10184969/

https://hyperledger-fabric.readthedocs.io/en/release-2.2/write_first_app.html

https://www.oracle.com/ie/blockchain/what-is-blockchain/blockchain-in-healthcare/

https://mui.com/material-ui/all-components/

https://blog.acviss.com/securing-your-brand-to-prevent-qr-code-duplication/

13. Appendix

- GitLab: https://gitlab.computing.dcu.ie/mahala2/2025-csc1097-mahala2-shahf3