Prof. M.Sheela Devi PESU-EC-CSE

to me ▾

Slides and LLD Approved

•••

--

Thanks & Regards,

Prof. M.Sheela Devi

Assistant Professor,

Department of Computer Science & Engineering,

PESU-EC Campus,

Hosur Road, Bengaluru

LOW LEVEL DESIGN AND IMPLEMENTATION DOCUMENT

BioBit: AI secured Supply Chain Management and Drug Authentication through Blockchain

Submitted by:

Vaani Bansal	**PES2UG21CS927**
Aditya Kumar Sinha	**PES2UG21CS035**
R Navaneeth Krishnan	**PES2UG21CS405**
Punith A	**PES2UG21CS403**

Under the guidance of

Prof Sheela Devi

Designation

PES University

August - December 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Bengaluru – 560 100, Karnataka, India

TABLE OF CONTENTS

1. Introduction

1.1. Overview

The Medicine Authenticity Verification System ensures end users that the medicine they are using is authentic. Users can scan the QR code on the medicine, and the system hashes the code and checks for its corresponding entry in Hyperledger. If an entry exists, the system confirms the medicine's authenticity. To secure the server, an LSTM model is used to monitor and detect any suspicious activity while the server queries Hyperledger for authenticity verification. The Hyperledger entries are maintained by a regulatory body that certifies all verified medicines.

1.2. Purpose

The purpose of this Low-Level Design (LLD) is to provide a detailed blueprint for the internal structure and workings of the Medicine Authenticity Verification System using Hyperledger and LSTM. This document elaborates on the system's module breakdown, class definitions, data flow mechanisms, and deployment architecture. By offering a granular view of each component, the LLD serves as a guide for developers to implement the system efficiently. It ensures that every part of the system, from QR code scanning and hashing, server querying of the Hyperledger, to LSTM-based anomaly detection for server protection, is well-defined, cohesive, and aligned with the project's security and performance objectives.

1.3. Scope

This low-level design outlines the functional modules of the Medicine Authenticity Verification System using Hyperledger and LSTM, including QR code scanning, Hyperledger-based verification, and server protection via LSTM. It describes the roles of each module and provides details of the associated classes, methods, and data members. By highlighting how these modules work together to ensure the authenticity of medicines and secure the server from anomalies, this document serves as a foundation for implementing and evaluating the system's performance, enhancing user trust in medicine authenticity and protecting the system from malicious activities.

2. Design Constraints, Assumptions, and Dependencies

2.1 Constraints:

1. Hyperledger query latency may cause delays in verifying medicine authenticity.
2. Ensuring real-time scanning and uploading of QR code data by end customers.
3. Data validation (keeping the Hyperledger entries updated with regulatory-approved medicine batches).
4. Scalability of the LSTM model for detecting anomalies as network traffic increases.

2.2 Assumptions:

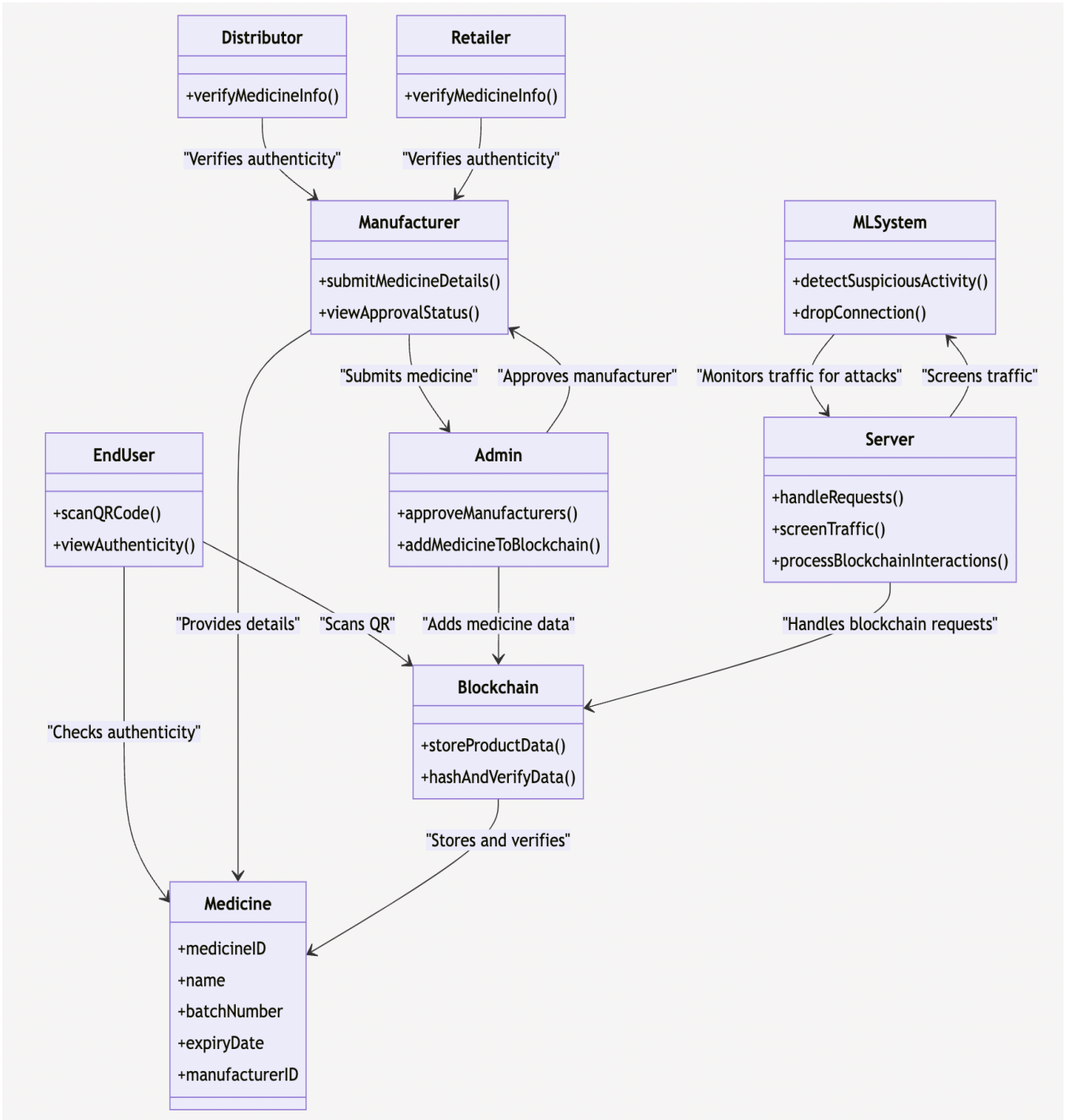
1. Access to up-to-date, validated medicine data from the regulatory body is uninterrupted.
2. Users have devices capable of scanning QR codes and uploading them for verification.
3. Users have reliable internet connectivity to query the Hyperledger for verification.

2.3 Dependencies:

1. Availability of regulatory-approved medicine data entries in Hyperledger.
2. Connectivity with external systems, including the Hyperledger blockchain for querying medicine authenticity.
3. Integration of the LSTM model for anomaly detection in real-time network traffic monitoring.

3. 1Design Description

3.1. Master Class Diagram



3.2. Modules

3.2.1. Description

3.2.1.1 Module 1: Blockchain-Based Medicine Authentication

This module focuses on ensuring the authenticity of medicines through the use of blockchain technology.

- **Admin**: The Admin verifies manufacturers and records approved medicine data (e.g., batch number, expiration date) on the Hyperledger blockchain. The data is immutable, ensuring that once recorded, it cannot be altered or tampered with.
- **User**: Users (such as consumers or retailers) scan the QR codes on medicine packages. The system queries the blockchain to verify the medicine's authenticity, returning confirmation if the medicine matches the blockchain data. Users can save the verification results for future reference.
- **Blockchain**: The blockchain stores medicine data securely, preventing the introduction of counterfeit products into the supply chain. Each medicine is hashed and stored with all associated details, ensuring transparency and reliability.

Key Features:

1. Immutable medicine records.
2. Secure verification through QR code scanning.
3. Admin-controlled entry of verified data.

3.2.1.2 Module 2: ML-Based Anomaly Detection for System Security

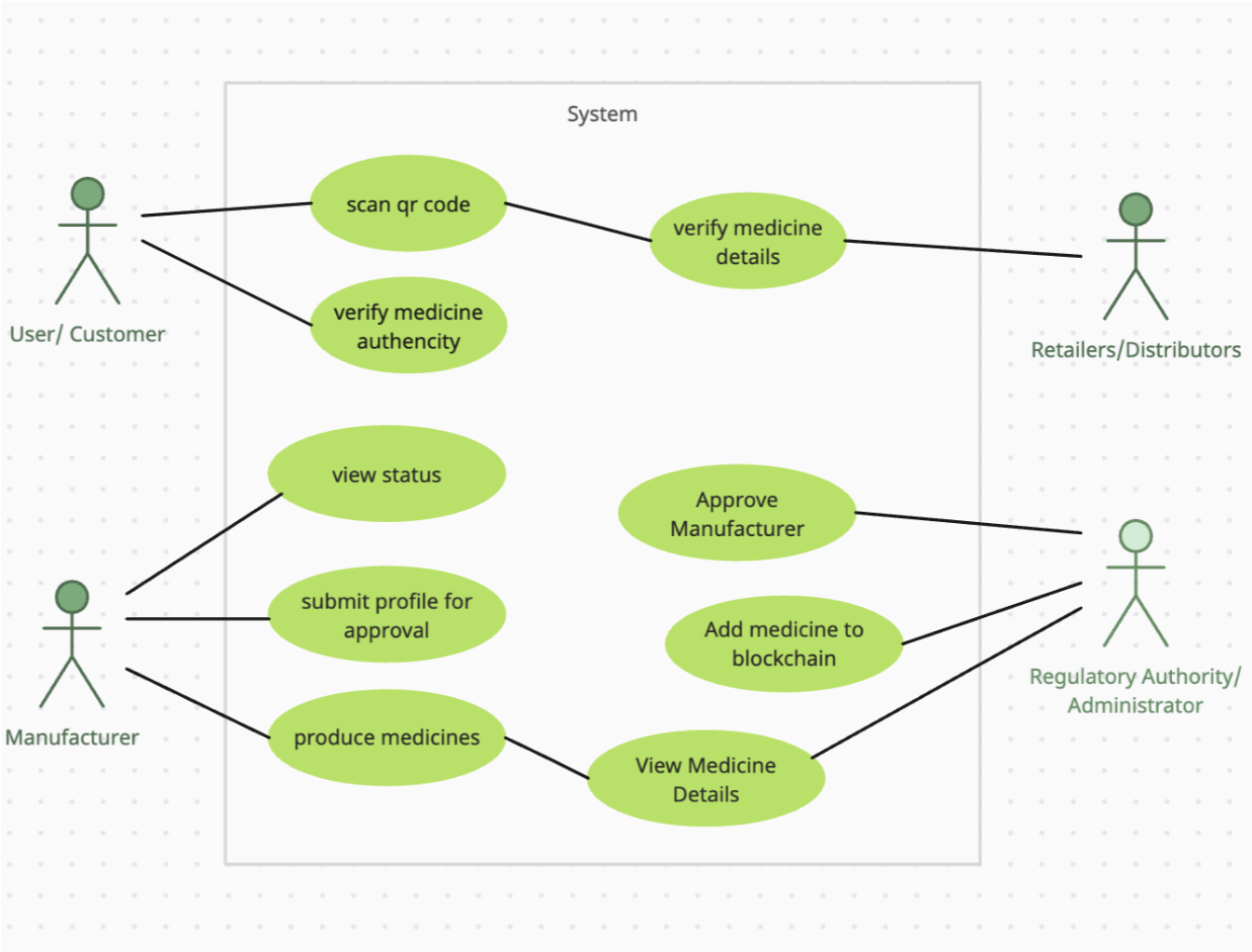
This module leverages machine learning to enhance the security of the system by detecting suspicious network activity.

- **LSTM Anomaly Detection**: An LSTM-based ML model continuously monitors network traffic for potential threats. If suspicious activity is detected, the system flags it and blocks the connection to prevent possible attacks.
- **Threat Prevention**: By screening requests, the ML model reduces the risk of attacks on the blockchain and ensures minimal downtime. It analyzes network data in real-time to detect anomalies that could indicate security breaches or cyberattacks.

Key Features:

1. Real-time anomaly detection.
2. Automated connection blocking to prevent attacks.
3. Continuous learning for improved threat detection.

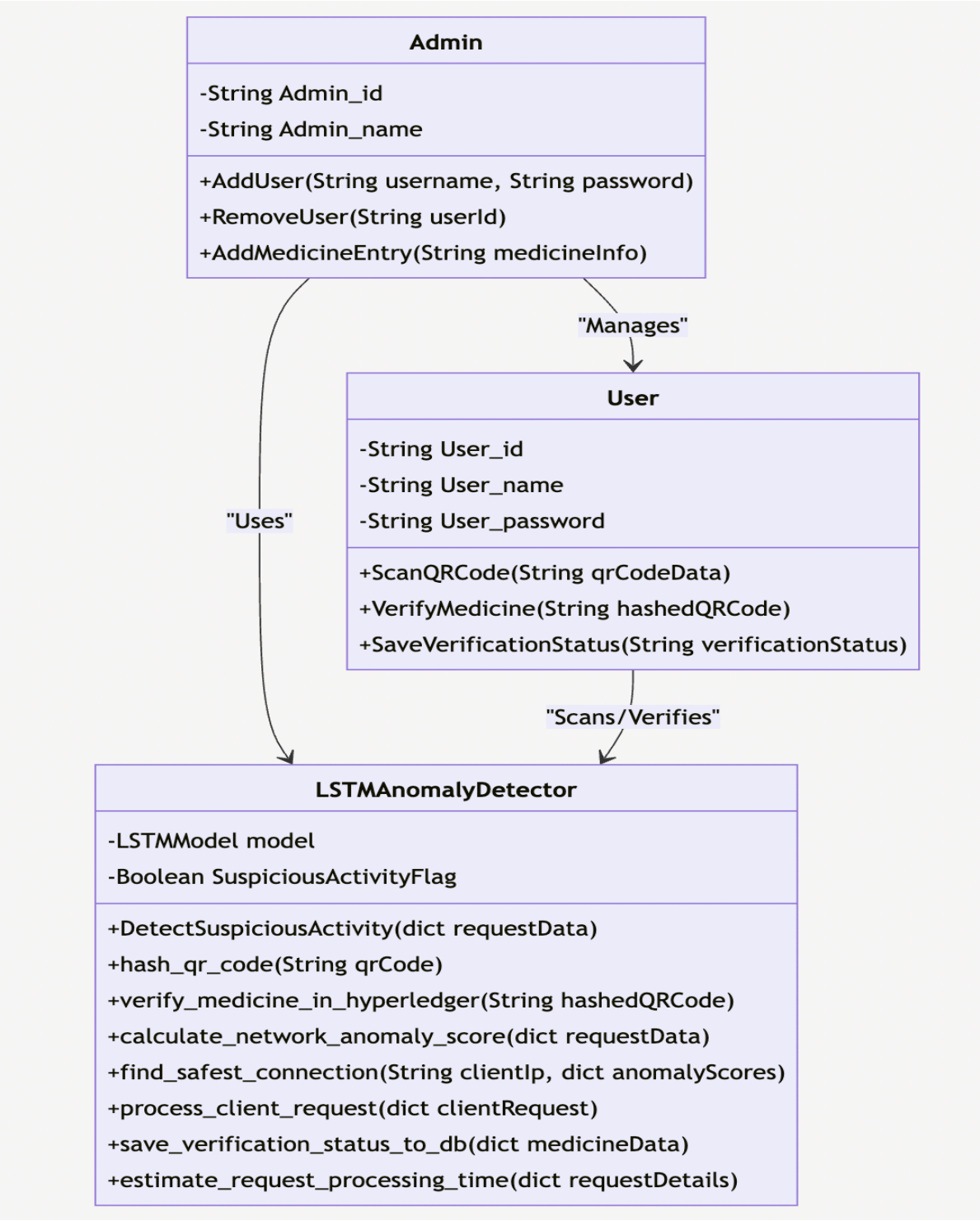
3.2.2. Use Case Diagram



3.2.3.

Use Case Item	Description
Scan QR Code	The end user scans the QR code on a medicine package via a front-end interface to check the authenticity of the medicine.
View Authenticity	The system retrieves relevant data from the blockchain and displays whether the medicine is authenticated or if there are any issues (e.g., counterfeit).
Submit Profile and medicine Details for approval	The manufacturer submits details of the medicine, including batch number, expiry date, and other relevant data, for approval by the admin.
Approve Manufacturer	The admin verifies and approves the manufacturer to ensure they are authorized to produce and distribute legitimate medicines that can be added to the blockchain
Add Medicine to Blockchain	The admin adds approved medicine details to the blockchain, where the information is stored securely and hashed to prevent tampering.
Verify Medicine Details	Distributors and retailers verify the authenticity of the medicine using data provided by the manufacturers.
View Status	Manufacturers can view the status of their approval from admin
Produce Medicines	Manufacturers have main responsibility of producing the medicines and then sending the details to admin for approval
View medicine Details	The admin or regulatory authority can view all the medicine details in order to approve them for adding to the blockchain.

3.2.4. Class Diagram:



3.2.4.1. Admin Class

3.2.4.2. Class Description

The Admin class represents the system administrator who verifies the manufacturer and then adds the medicine onto the blockchain if the medicine produced is legitimate,else flags the manufacturer/medicine. Only they have access to the hyperledger.

3.2.4.3. Data members

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	Admin_id	Private	None	Unique identifier for the admin
String	Admin_name	Private	None	Name of the admin

3.2.4.4. Method 1: `AddUser()`

- **Purpose**: Add a new user to the system.
- **Input**: User details (e.g., username, password).
- **Output**: Success/Failure status of user addition.
- **Parameters**: User information (e.g., username, password).
- **Exceptions**: `DuplicateUserException`, `InvalidInputException`.

.

if (user does not exist in system)

create new User object
add User to database
return success
else
throw DuplicateUserException

3.2.4.5. Method 2: RemoveUser()

- Purpose: Remove an existing user from the system
- Input: User ID
- Output: Success/Failure status of user removal
- Parameters: User ID
- Exceptions: UserNotFoundException
- Pseudo-code:

if (user exists in system)
remove User from
database
return success
else
throw UserNotFoundException

3.2.4.6. Method 3: AddMedicineEntry()

- Purpose: Add, remove, or update medicine entries in the Hyperledger blockchain.
- Input: Medicine details (e.g., medicine batch, regulatory approval).
- Output: Success/Failure status of data update.
- Parameters: Medicine information (e.g., batch number, expiration date, regulatory approval status).
- Exceptions: InvalidSourceException.
- Pseudo Code:

if medicine entry is valid:
add/update/remove medicine entry in blockchain
return success
else:
throw InvalidSourceException

3.2.4.7. User Class

The User class represents a system user who can scan QR codes, verify medicine authenticity, and manage their account..

3.2.4.8. Data Members :

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	User_id	Private	None	Unique identifier for the user
String	User_name	Private	None	Name of the user
String	User_password	Private	None	Encrypted password of the user

3.2.4.9. Method 1: `ScanQRCode()`

- **Purpose**: Allows the user to scan and upload the QR code of the medicine.
- **Input**: QR code data.
- **Output**: Success/Failure status.
- **Parameters**: QR code data as string.
- **Exceptions**: `InvalidQRCodeException`.
- **Pseudo-code**:

```
python
Copy code
if QR code is valid:
    hash the QR code
    send hashed code to Hyperledger for verification
    return success
else:
    throw InvalidQRCodeException
```

3.2.4.10. Method 2: `VerifyMedicine()`

- **Purpose**: Query the Hyperledger blockchain to verify the authenticity of the scanned medicine.
- **Input**: Hashed QR code.
- **Output**: Verification result (authentic/not authentic).
- **Parameters**: Hashed QR code data.
- **Exceptions**: `MedicineNotFoundException`.
- **Pseudo-code**:

```
python
Copy code
if medicine entry is found in Hyperledger:
    return "Medicine is authentic"
else:
    throw MedicineNotFoundException
```

3.2.4.11. `Method 3: SaveVerificationStatus()`

- **Purpose**: Save the verification result for future reference.
- **Input**: Verification status (authentic/not authentic).
- **Output**: Success/Failure status.
- **Parameters**: Verification status data.
- **Exceptions**: `StorageException`.
- **Pseudo-code**:

```
python
Copy code
try:
    save the verification status to user's history
    return success
catch StorageException:
    throw StorageException
```

3.2.4.12. LSTM Anomaly Detector Class:

The **LSTM Anomaly Detector** class is responsible for detecting suspicious activity in the network while the system processes client requests to the Hyperledger blockchain.

3.2.4.13. Data Members:

Data Type	Data Name	Access Modifiers	Initial Value	Description
LSTMModel	model	Private	Pretrained LSTM model	The LSTM model for anomaly detection
Boolean	SuspiciousActivityFlag	Private	False	Flag indicating if suspicious activity is detected

3.2.4.14. Method 1: `DetectSuspiciousActivity()`

- **Purpose**: Detect suspicious activity in the network using the LSTM model.
- **Input**: Network packet data.
- **Output**: Suspicious/Normal status.
- **Parameters**: Network traffic data.

- **Exceptions**: None.
- **Pseudo-code**:

```
python
Copy code
for each incoming request:
    if LSTM model predicts suspicious activity:
        return "Suspicious"
    else:
        return "Normal"
```

a. `hash_qr_code(qr_code)`

- **Purpose**: Hashes the QR code of a medicine to ensure the security of its verification process.
- **Input**:
  - `qr_code`: The string representation of the QR code.
- **Output**: A hashed string representing the QR code.
- **Parameters**: `(qr_code: str)`
- **Exceptions**: Throws an exception if the QR code is invalid or empty.
- **Pseudo-code**:

```
python
Copy code
if qr_code is valid:
    hashed_qr = hash_function(qr_code)
    return hashed_qr
else:
    throw InvalidQRCodeException
```

b. `verify_medicine_in_hyperledger(hashed_qr_code)`

- **Purpose**: Verifies the authenticity of the medicine by checking the hashed QR code against the Hyperledger blockchain.
- **Input**:
  - `hashed_qr_code`: Hashed string generated from the QR code.
- **Output**: A boolean indicating whether the medicine is authentic (True) or counterfeit (False).
- **Parameters**: `(hashed_qr_code: str)`
- **Exceptions**: Throws an exception if no entry is found in Hyperledger.
- **Pseudo-code**:

```
python
Copy code
result = query_hyperledger(hashed_qr_code)
if result exists:
    return True # Medicine is authentic
else:
    throw MedicineNotFoundException
```

c. `calculate_network_anomaly_score(request_data)`

- **Purpose**: Calculates the anomaly score of a network request using an LSTM model

to detect suspicious activity.

- **Input**:
  - `request_data`: The network request data.
- **Output**: A score indicating the likelihood of the request being suspicious.
- **Parameters**: `(request_data: dict)`
- **Exceptions**: Throws an exception if the request data is incomplete.

- **Pseudo-code**:

```
python
Copy code
if request_data is valid:
    anomaly_score = lstm_model.predict(request_data)
    return anomaly_score
else:
    throw InvalidRequestDataException
```

d. `find_safest_connection(client_ip, anomaly_scores)`

- **Purpose**: Finds the connection with the lowest anomaly score from a list of connections, ensuring a secure connection for the client.
- **Input**:
  - `client_ip`: The IP address of the client.
  - `anomaly_scores`: A dictionary mapping network connections to their respective anomaly scores.
- **Output**: The connection with the lowest anomaly score.
- **Parameters**: `(client_ip: str, anomaly_scores: dict)`
- **Exceptions**: Returns `None` if no safe connections are found.
- **Pseudo-code**:

```
python
Copy code
if client_ip in anomaly_scores:
    lowest_score_connection = min(anomaly_scores,
key=anomaly_scores.get)
    return lowest_score_connection
else:
    return None
```

e. `process_client_request(client_request)`

- **Purpose**: Processes a client's request to verify a medicine by checking the Hyperledger and assessing network security using LSTM.
- **Input**:
  - `client_request`: A dictionary containing the client's request, including the scanned QR code and network data.
- **Output**: A response indicating whether the medicine is authentic or suspicious.
- **Parameters**: `(client_request: dict)`
- **Exceptions**: Throws an exception if the request is invalid or if an anomaly is detected.
- **Pseudo-code**:

```
python
Copy code
qr_code = client_request['qr_code']
request_data = client_request['request_data']

hashed_qr = hash_qr_code(qr_code)
```

```
anomaly_score = calculate_network_anomaly_score(request_data) if
anomaly_score > threshold:
    throw SuspiciousActivityException

if verify_medicine_in_hyperledger(hashed_qr):
    return "Medicine is authentic"
else:
    throw MedicineNotFoundException
```

f. `save_verification_status_to_db(medicine_data)`

- **Purpose**: Saves the verification status of a medicine in the database for future reference and auditing.
- **Input**:
  - `medicine_data`: A dictionary containing the details of the verified medicine.
- **Output**: Success/Failure status.
- **Parameters**: `(medicine_data: dict)`
- **Exceptions**: Throws an exception if the data cannot be saved.
- **Pseudo-code**:

```
python
Copy code
try:
    save medicine_data to database
    return success
except DatabaseException:
    throw StorageException
```

g. `estimate_request_processing_time(request_details)`

- **Purpose**: Estimates the time taken to process a client request for verifying a medicine, including network latency and blockchain query time.
- **Input**:
  - `request_details`: The details of the client request, including the QR code and network parameters.
- **Output**: Estimated processing time in milliseconds.
- **Parameters**: `(request_details: dict)`
- **Exceptions**: None.
- **Pseudo-code**:

```
python
Copy code
processing_time = calculate_query_time(request_details['qr_code'])
network_latency =
estimate_network_latency(request_details['network_data'])
total_time = processing_time + network_latency
return total_time
```