Task-2: Blockchain-based Product Verification using QR Code

Objective:

The objective of Task-2 is to implement a blockchain-based product verification system that uses QR codes for authenticating products by comparing the product hash stored in the blockchain and the database.

Technologies Used:

Blockchain: Polygon (Amoy Testnet)
Smart Contract: Solidity
Backend: Node.js, Express.js
Database: PostgreSQL
QR Code Generation: QRCode.js

System Architecture:

Frontend: A simple web interface to input product details and fetch QR codes.
Backend: Node.js server handling interactions between the blockchain and the database.
Blockchain: Smart contracts to store and retrieve product details and hashes.

Steps Involved:

Step 1: Smart Contract Implementation

Creating a contract to add product data to the blockchain and retrieve it.

Smart Contract Code:

```
pragma solidity ^0.8.0;

contract ProductVerification {
  struct Product {
    uint256 p_id;
    string p_name;
    string p_m_date;
    string p_batch;
    string p_hash;
  }
```

```
    mapping(uint256 => Product) public products;

    function addData(
        uint256 p_id,
        string memory p_name,
        string memory p_m_date,
        string memory p_batch,
        string memory p_hash
    ) public {
        products[p_id] = Product(p_id, p_name, p_m_date, p_batch, p_hash);
    }

    function getData(uint256 p_id) public view returns (
        uint256,
        string memory,
        string memory,
        string memory,
        string memory
    ) {
        Product memory product = products[p_id];
        return (product.p_id, product.p_name, product.p_m_date, product.p_batch,
product.p_hash);
    }
}
```

Step 2: Backend API for Storing and Retrieving Data

Node.js server to interact with the blockchain and store data in the database.

API Code for Saving and Verifying Product Details:

```
app.post('/api/data', async (req, res) => {
    const { p_id, p_name, p_m_date, p_batch } = req.body;
    const hash = crypto.createHash('sha256').update(p_id + p_name + p_m_date +
p_batch).digest('hex');
    // Blockchain interaction code to store data
    const transactionData = mfg_contract.methods.addData(p_id, p_name, p_m_date,
p_batch, hash).encodeABI();
    // Code for sending transaction to blockchain...
    await client.query(insertQuery, [p_id, p_name, p_m_date, hash, qrCodeBase64]);
});

app.get('/api/getQR/:id', async (req, res) => {
    const p_id = req.params.id;
    // Fetch QR code from database and send it as response
```

```
    const query = 'SELECT qr_code FROM product_data WHERE product_id = $1';
    const result = await client.query(query, [p_id]);
    res.json({ qrCodeText: result.rows[0].qr_code });
  });

  app.post('/api/verify', async (req, res) => {
    const { p_id, p_qr } = req.body;
    // Decode QR and compare with blockchain and database hashes
    const decodedHash = decodeQRCode(p_qr);
    const blockchainData = await mfg_contract.methods.getData(p_id).call();
    const dbData = await client.query('SELECT product_hash FROM product_data WHERE
product_id = $1', [p_id]);

    if (decodedHash === blockchainData.p_hash && decodedHash ===
dbData.rows[0].product_hash) {
      res.json({ message: 'Authentic Product' });
    } else {
      res.json({ message: 'Product Not Authentic' });
    }
  });
```

Steps for QR Code Generation and Validation:

Generate QR Code: Use the QRCode library to generate a QR code from the product hash.
Validate QR Code: Decode the QR code and compare it with the product hash from both the blockchain and the database.

Conclusion:

The task was completed successfully, ensuring that products can be verified using blockchain-based data and QR code technology. The solution ensures that product information is secure and easily verifiable.